



# *Developing Palm OS 3.0 Applications*

## **Part I: Interface Management**

**Navigate this online document as follows:**

---

To see bookmarks,  
type:

Command-7 (Mac OS)  
Ctrl-7 (Windows)

---

To navigate,  
click on:

any blue hypertext link  
any [Table of Contents](#) entry  
any [Index](#) entry  
arrows in the toolbar

---



# **Developing Palm OS 3.0 Applications**

## **Part I: Interface Management**

Copyright © 1996 - 1998, 3Com Corporation or its subsidiaries ("3Com"). All rights reserved. This documentation may be printed and copied solely for use in developing products for the Palm Computing platform. 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 3Com.

3Com reserves the right to revise this documentation and to make changes in content from time to time without obligation on the part of 3Com to provide notification of such revision or changes. 3COM MAKES 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. 3COM MAKES 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, 3COM 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 3COM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

3Com, the 3Com logo, HotSync, Palm Computing, and Graffiti are registered trademarks, and Palm III, Palm OS, and the Palm Computing Platform logo are trademarks of 3Com Corporation or its subsidiaries.

Microsoft and Windows are registered trademarks of Microsoft Corporation. Other brand and product names may be registered trademarks or trademarks of their respective holders.

IF THIS DOCUMENTATION IS PROVIDED ON A COMPACT DISK, THE OTHER SOFTWARE AND DOCUMENTATION ON THE COMPACT DISK ARE SUBJECT TO THE LICENSE AGREEMENT ACCOMPANYING THE COMPACT DISK.

## Contact Information:

---

<b>Metrowerks U.S.A. and international</b>	Metrowerks Corporation 2201 Donley Drive, Suite 310 Austin, TX 78758 U.S.A.
<b>Metrowerks Canada</b>	Metrowerks Inc. 1500 du College, Suite 300 Ville St-Laurent, QC Canada H4L 5G6
<b>Metrowerks Mail order</b>	Voice: 1-800-377-5416 Fax: 1-512-873-4901
<b>3Com (Palm Computing Subsidiary) Mail Order</b>	U.S.A.: 1-800-881-7256    Canada: 800-891-6342 elsewhere: 1-801-431-1536
<b>Metrowerks World Wide Web</b>	<a href="http://www.metrowerks.com">http://www.metrowerks.com</a>
<b>Palm Computing World Wide Web</b>	<a href="http://www.palm.com">http://www.palm.com</a>
<b>Registration information</b>	<a href="mailto:register@metrowerks.com">register@metrowerks.com</a>
<b>Technical support</b>	<a href="mailto:support@metrowerks.com">support@metrowerks.com</a>
<b>Sales, marketing, &amp; licensing</b>	<a href="mailto:sales@metrowerks.com">sales@metrowerks.com</a>
<b>CompuServe</b>	go Metrowerks

---

# Table of Contents

---

<b>About This Document.</b> . . . . .	<b>23</b>
Palm OS SDK Documentation . . . . .	23
What This Guide Contains . . . . .	24
Conventions Used in This Guide . . . . .	25
What's New in Palm OS 3.0 . . . . .	26
General Information . . . . .	26
New Launch Codes. . . . .	27
Dynamic User Interface Objects . . . . .	27
Font Functions . . . . .	27
Progress Manager . . . . .	28
File Streaming API . . . . .	28
Sound Manager . . . . .	29
Exchange Manager . . . . .	29
IR Library . . . . .	29
Miscellaneous New Functions in 3.0 . . . . .	30
Existing Functions that Changed in 3.0 . . . . .	30
Documentation Revisions . . . . .	31
<b>1 Developing Palm OS Applications</b> . . . . .	<b>33</b>
Overview of Application Development . . . . .	33
Designing UI and Program Functionality . . . . .	34
Constructing UI Resources . . . . .	35
Using Managers and Filling Out the Program Logic. . . . .	36
Writing Robust Code . . . . .	37
Building, Debugging, and Testing . . . . .	40
Building the Application and Running it on the Palm Device. . . . .	40
Using Other Components of the SDK . . . . .	41
Internal Structure of an Application . . . . .	41
The 'code' #1 Resource . . . . .	42
The 'pref' #0 Resource . . . . .	42
The 'code' #0 and 'date' #0 Resources . . . . .	43
Naming Conventions . . . . .	43
Basic Hardware . . . . .	44
RAM and ROM . . . . .	44

## Table of Contents

---

Modes of Operation . . . . .	44
Palm OS Connectivity. . . . .	45
Real-Time Clock and Timer . . . . .	46
Palm OS Device Screen and Sound Generation . . . . .	46
Palm OS Device Reset Switch . . . . .	47
Different Palm Computing Platform Devices . . . . .	48
Running Older Applications on the 3.0 Device . . . . .	49
Compiling Older Applications With SDK Version 3.0 . . . . .	49
Using OS Version 3.0 Features . . . . .	50
Running 3.0 Applications on an Older Device . . . . .	50
Retrieving the System Version Number . . . . .	51
Retrieving the ROM Serial Number. . . . .	51
<b>2 Application Control Flow . . . . .</b>	<b>55</b>
How Events Control an Application. . . . .	57
Basic Application Stages. . . . .	57
The Startup Routine . . . . .	58
The Event Loop . . . . .	61
The Stop Routine. . . . .	65
How Launch Codes Control an Application . . . . .	66
Parameter Block . . . . .	67
Launch Flags . . . . .	67
Launch Code Example . . . . .	68
Summary of All Launch Codes . . . . .	70
More About Launch Codes. . . . .	71
sysAppLaunchCmdAlarmTriggered . . . . .	72
sysAppLaunchCmdCountryChange . . . . .	72
sysAppLaunchCmdDisplayAlarm . . . . .	72
sysAppLaunchCmdExgAskUser . . . . .	72
sysAppLaunchCmdExgReceiveData . . . . .	74
sysAppLaunchCmdFind . . . . .	74
sysAppLaunchCmdGoto . . . . .	76
sysAppLaunchCmdInitDatabase . . . . .	77
sysAppLaunchCmdLookup . . . . .	78
sysAppLaunchCmdPanelCalledFromApp. . . . .	79
sysAppLaunchCmdReturnFromPanel . . . . .	80

sysAppLaunchCmdSaveData . . . . .	80
sysAppLaunchCmdSyncNotify . . . . .	80
sysAppLaunchCmdSystemLock . . . . .	81
sysAppLaunchCmdSystemReset . . . . .	81
sysAppLaunchCmdTimeChange . . . . .	82
More About Launch Flags . . . . .	82
Responding to Launch Codes . . . . .	83
Determining Status When Receiving Launch Code . . . . .	84
Predefined Launch Codes . . . . .	84
Creating Your Own Launch Codes . . . . .	84
<b>3 Palm OS User Interface Resources. . . . .</b>	<b>85</b>
Using Constructor to Work With Resources . . . . .	86
Creating Resources . . . . .	86
Changing Resources . . . . .	87
Project Resources . . . . .	88
Alerts . . . . .	89
Form Resource . . . . .	91
String Resource. . . . .	94
Menus and Menu Bars . . . . .	94
Menu Overview . . . . .	94
Creating a Menu . . . . .	95
Menu Bar and Menu Resources . . . . .	96
Menu User Interaction . . . . .	96
Event Flow for Menu Resource. . . . .	98
Catalog Resources . . . . .	98
Button Resource . . . . .	100
Check Box Resource. . . . .	103
Field Resource . . . . .	106
Form Bitmap Resource . . . . .	109
Gadget Resource . . . . .	110
Graffiti Shift Indicator Resource . . . . .	111
Label Resource . . . . .	112
List Resource. . . . .	113
Popup Trigger Resource . . . . .	115
Push Button Resource . . . . .	117

## Table of Contents

---

Repeating Button Resource . . . . .	120
Scrollbar Resource . . . . .	122
Selector Trigger Resource . . . . .	124
Table Resource . . . . .	126
<b>4 Palm OS User Interface Objects . . . . .</b>	<b>127</b>
A Note on the Rectangle Structure . . . . .	128
Control Objects. . . . .	128
Control Object Events. . . . .	129
Structure of a Control . . . . .	130
Associated Resources . . . . .	133
Control Functions . . . . .	133
Date and Time Objects. . . . .	134
Date and Time Functions . . . . .	134
Field Objects . . . . .	134
Field Object Events . . . . .	135
Structure of a Field . . . . .	136
Associated Resources . . . . .	139
Field Functions . . . . .	140
Form Objects. . . . .	142
Form Object Events. . . . .	142
Structure of a Form. . . . .	144
Associated Resource . . . . .	149
Form Functions . . . . .	149
Insertion Point Object . . . . .	151
Insertion Point Functions . . . . .	151
List Object . . . . .	152
List Object Events . . . . .	153
Structure of a List . . . . .	154
Associated Resources . . . . .	156
List Functions . . . . .	156
Menu Objects . . . . .	157
Menu Events . . . . .	158
Structure of a Menu . . . . .	158
Menu Functions . . . . .	162
Scrollbar Object. . . . .	162

Table Objects . . . . .	166
Table Event . . . . .	166
Structure of a Table . . . . .	166
Associated Resource . . . . .	169
Table Functions . . . . .	169
Window Objects . . . . .	171
Window Events . . . . .	171
Structure of a Window . . . . .	171
Window Functions . . . . .	173
Dynamic User Interface Objects. . . . .	175
Dynamic User Interface Functions . . . . .	176
<b>5 Using Palm OS UI Managers . . . . .</b>	<b>179</b>
The Alert Manager . . . . .	180
Alert Resource Information . . . . .	180
Alert Manager Functions . . . . .	181
The Graffiti Manager . . . . .	181
Using GrfProcessStroke . . . . .	181
Using Other High-Level Graffiti Manager Calls . . . . .	182
Special-Purpose Graffiti Manager Calls . . . . .	182
Accessing Graffiti ShortCuts . . . . .	182
Note on Auto Shifting. . . . .	183
Note on Graffiti Help . . . . .	183
Graffiti Manager Functions . . . . .	183
The Key Manager. . . . .	184
The Pen Manager . . . . .	185
The Progress Manager. . . . .	186
Progress textCallback Function. . . . .	186
Progress Manager Function Summary. . . . .	189
<b>6 Palm OS Events . . . . .</b>	<b>191</b>
appStopEvent . . . . .	192
ctlEnterEvent . . . . .	192
ctlExitEvent . . . . .	192
ctlRepeatEvent . . . . .	193
ctlSelectEvent . . . . .	193

## Table of Contents

---

daySelectEvent . . . . .	194
fldChangedEvent . . . . .	194
fldEnterEvent . . . . .	195
fldHeightChangedEvent. . . . .	195
frmCloseEvent . . . . .	195
frmGotoEvent . . . . .	196
frmLoadEvent . . . . .	197
frmOpenEvent . . . . .	197
frmSaveEvent . . . . .	197
frmUpdateEvent . . . . .	197
frmTitleEnterEvent . . . . .	198
frmTitleSelectEvent . . . . .	198
keyDownEvent. . . . .	199
lstEnterEvent. . . . .	200
lstExitEvent . . . . .	200
lstSelectEvent . . . . .	200
menuEvent . . . . .	201
nilEvent . . . . .	201
penDownEvent. . . . .	202
penMoveEvent . . . . .	202
penUpEvent . . . . .	202
popSelectEvent. . . . .	203
sclEnterEvent . . . . .	204
sclExitEvent . . . . .	204
sclRepeatEvent . . . . .	205
tblEnterEvent . . . . .	205
tblExitEvent . . . . .	206
tblSelectEvent . . . . .	206
winEnterEvent . . . . .	207
winExitEvent . . . . .	207
<b>7 Palm OS User Interface Functions . . . . .</b>	<b>209</b>
Category Functions . . . . .	209
CategoryCreateList . . . . .	209
CategoryCreateListV10 . . . . .	210
<u>CategoryEdit</u> . . . . .	<u>211</u>

## Table of Contents

---

<a href="#"><u>CategoryEditV20</u></a>	<a href="#"><u>212</u></a>
CategoryEditV10 . . . . .	213
CategoryFind . . . . .	213
CategoryFreeList . . . . .	214
CategoryFreeListV10 . . . . .	215
CategoryGetName . . . . .	215
CategoryGetNext. . . . .	216
CategoryInitialize . . . . .	216
CategorySelect . . . . .	217
CategorySelectV10 . . . . .	218
CategorySetName . . . . .	219
CategorySetTriggerLabel . . . . .	219
CategoryTruncateName . . . . .	220
Clipboard Functions . . . . .	220
ClipboardAddItem . . . . .	220
ClipboardGetItem . . . . .	221
Control Functions. . . . .	221
CtlDrawControl . . . . .	221
CtlEnabled . . . . .	222
CtlEraseControl . . . . .	222
CtlGetLabel . . . . .	223
CtlGetValue . . . . .	223
CtlHandleEvent . . . . .	224
CtlHideControl . . . . .	225
CtlHitControl . . . . .	225
<a href="#"><u>CtlNewControl</u></a>	<a href="#"><u>226</u></a>
CtlSetEnabled . . . . .	228
CtlSetLabel . . . . .	228
CtlSetUsable. . . . .	229
CtlSetValue . . . . .	229
CtlShowControl . . . . .	230
<a href="#"><u>CtlValidatePointer</u></a>	<a href="#"><u>230</u></a>
Field UI Functions . . . . .	231
FldCalcFieldHeight. . . . .	231
FldCompactText . . . . .	231
FldCopy . . . . .	232

## Table of Contents

---

FldCut . . . . .	232
FldDelete . . . . .	233
FldDirty. . . . .	233
FldDrawField . . . . .	234
FldEraseField . . . . .	234
FldFreeMemory . . . . .	235
FldGetAttributes . . . . .	235
FldGetBounds . . . . .	236
FldGetFont . . . . .	236
FldGetInsPtPosition . . . . .	237
FldGetMaxChars . . . . .	237
FldGetNumberOfBlankLines . . . . .	238
FldGetScrollPosition . . . . .	238
FldGetScrollValues . . . . .	239
FldGetSelection . . . . .	240
FldGetTextAllocatedSize . . . . .	241
FldGetTextHandle . . . . .	241
FldGetTextHeight . . . . .	242
FldGetTextLength . . . . .	242
FldGetTextPtr . . . . .	243
FldGetVisibleLines . . . . .	243
FldGrabFocus . . . . .	244
FldHandleEvent . . . . .	245
FldInsert . . . . .	246
FldMakeFullyVisible . . . . .	246
<a href="#"><u>FldNewField</u></a> . . . . .	<a href="#"><u>247</u></a>
FldPaste. . . . .	249
FldRecalculateField. . . . .	250
FldReleaseFocus . . . . .	250
FldScrollable. . . . .	251
FldScrollField . . . . .	251
FldSendChangeNotification . . . . .	252
FldSendHeightChangeNotification . . . . .	252
FldSetAttributes . . . . .	253
FldSetBounds . . . . .	253
FldSetDirty . . . . .	254

## Table of Contents

---

FldSetFont. . . . .	254
FldSetInsertionPoint . . . . .	255
FldSetInsPtPosition. . . . .	255
FldSetMaxChars . . . . .	256
FldSetScrollPosition . . . . .	256
FldSetSelection. . . . .	257
FldSetText. . . . .	258
FldSetTextAllocatedSize. . . . .	259
FldSetTextHandle . . . . .	259
FldSetTextPtr . . . . .	260
FldSetUsable . . . . .	260
FldUndo . . . . .	261
FldWordWrap . . . . .	261
Font Functions . . . . .	262
New Font Features in Palm OS 3.0 . . . . .	262
FntAverageCharWidth . . . . .	263
FntBaseLine . . . . .	263
FntCharHeight. . . . .	263
<u>FntCharsInWidth</u> . . . . .	<u>264</u>
<u>FntCharsWidth</u> . . . . .	<u>265</u>
<u>FntCharWidth</u> . . . . .	<u>265</u>
<u>FntDefineFont</u> . . . . .	<u>266</u>
<u>FntDescenderHeight</u> . . . . .	<u>267</u>
FntGetFont . . . . .	267
FntGetFontPtr . . . . .	267
<u>FntGetScrollValues</u> . . . . .	<u>268</u>
FntLineHeight . . . . .	268
<u>FntLineWidth</u> . . . . .	<u>269</u>
FntSetFont . . . . .	269
<u>FntWordWrap</u> . . . . .	<u>270</u>
<u>FntWordWrapReverseNLines</u> . . . . .	<u>270</u>
<u>FontSelect</u> . . . . .	<u>271</u>
Form Functions. . . . .	272
FrmAlert . . . . .	272
FrmCloseAllForms . . . . .	272
FrmCopyLabel. . . . .	273
FrmCopyTitle . . . . .	274

## Table of Contents

---

FrmCustomAlert . . . . .	275
FrmDeleteForm . . . . .	276
FrmDispatchEvent . . . . .	276
FrmDoDialog . . . . .	277
FrmDrawForm. . . . .	277
FrmEraseForm. . . . .	278
FrmGetActiveForm. . . . .	278
FrmGetActiveFormID. . . . .	279
FrmGetControlGroupSelection. . . . .	279
FrmGetControlValue . . . . .	280
FrmGetFirstForm. . . . .	280
FrmGetFocus . . . . .	281
FrmGetFormBounds . . . . .	281
FrmGetFormId. . . . .	282
FrmGetFormPtr . . . . .	282
FrmGetGadgetData. . . . .	283
FrmGetLabel . . . . .	283
FrmGetNumberOfObjects . . . . .	284
FrmGetObjectBounds. . . . .	284
FrmGetObjectId . . . . .	285
FrmGetObjectIndex. . . . .	285
FrmGetObjectPosition . . . . .	286
FrmGetObjectPtr . . . . .	286
FrmGetObjectType . . . . .	287
FrmGetTitle . . . . .	287
FrmGetUserModifiedState. . . . .	288
FrmGetWindowHandle . . . . .	288
FrmGotoForm . . . . .	289
FrmHandleEvent. . . . .	289
FrmHelp . . . . .	290
FrmHideObject . . . . .	290
FrmInitForm. . . . .	291
<a href="#"><u>FrmNewBitmap</u></a> . . . . .	<a href="#"><u>292</u></a>
<a href="#"><u>FrmNewForm</u></a> . . . . .	<a href="#"><u>293</u></a>
<a href="#"><u>FrmNewGadget</u></a> . . . . .	<a href="#"><u>295</u></a>
<a href="#"><u>FrmNewLabel</u></a> . . . . .	<a href="#"><u>296</u></a>

## Table of Contents

---

FrmPointInTitle . . . . .	297
FrmPopupForm . . . . .	297
<u>FrmRemoveObject</u> . . . . .	<u>298</u>
FrmReturnToForm . . . . .	299
FrmSaveAllForms . . . . .	299
FrmSetActiveForm . . . . .	300
FrmSetCategoryLabel . . . . .	300
FrmSetControlGroupSelection . . . . .	301
FrmSetControlValue . . . . .	301
FrmSetEventHandler . . . . .	302
FrmSetFocus . . . . .	302
FrmSetGadgetData . . . . .	303
FrmSetMenu . . . . .	303
FrmSetNotUserModified . . . . .	304
FrmSetObjectBounds . . . . .	304
FrmSetObjectPosition . . . . .	305
FrmSetTitle . . . . .	306
FrmShowObject . . . . .	306
FrmUpdateForm . . . . .	307
FrmUpdateScrollers . . . . .	307
<u>FrmValidatePtr</u> . . . . .	<u>308</u>
FrmVisible . . . . .	308
Character Attribute Functions . . . . .	309
GetCharAttr . . . . .	309
GetCharCaselessValue . . . . .	309
GetCharSortValue . . . . .	310
Graffiti Manager Functions . . . . .	311
GrfAddMacro . . . . .	311
GrfAddPoint . . . . .	311
GrfCleanState . . . . .	312
GrfDeleteMacro . . . . .	312
GrfFilterPoints . . . . .	313
GrfFindBranch . . . . .	313
GrfFlushPoints . . . . .	314
GrfGetAndExpandMacro . . . . .	314
GrfGetGlyphMapping . . . . .	315

## Table of Contents

---

GrfGetMacro . . . . .	316
GrfGetMacroName . . . . .	316
GrfGetNumPoints . . . . .	317
GrfGetPoint . . . . .	317
GrfGetState . . . . .	318
GrfInitState . . . . .	319
GrfMatch . . . . .	319
GrfMatchGlyph . . . . .	320
GrfProcessStroke . . . . .	321
GrfSetState . . . . .	322
SysGrfShortCutListDialog . . . . .	322
Functions for System Use Only . . . . .	323
GraffitiShift Functions . . . . .	323
GsiEnable . . . . .	323
GsiEnabled . . . . .	324
GsiInitialize . . . . .	324
GsiSetLocation . . . . .	324
GsiSetShiftState . . . . .	325
Insertion Point Functions . . . . .	326
InsPtEnable . . . . .	326
InsPtEnabled . . . . .	326
InsPtGetHeight . . . . .	327
InsPtGetLocation . . . . .	327
InsPtSetHeight . . . . .	328
InsPtSetLocation . . . . .	328
Functions for System Use Only . . . . .	329
Key Manager Functions . . . . .	329
KeyCurrentState . . . . .	329
KeyRates . . . . .	330
Functions for System Use Only . . . . .	330
List UI Functions . . . . .	332
LstDrawList . . . . .	332
LstEraseList . . . . .	332
LstGetNumberOfItems . . . . .	333
LstGetSelection . . . . .	333
LstGetSelectionText . . . . .	334

## Table of Contents

---

LstGetVisibleItems . . . . .	334
LstHandleEvent . . . . .	335
LstMakeItemVisible . . . . .	336
<u>LstNewList</u>	<u>336</u>
LstPopupList . . . . .	338
LstScrollList . . . . .	338
LstSetDrawFunction . . . . .	339
LstSetHeight. . . . .	339
LstSetListChoices . . . . .	340
LstSetPosition . . . . .	340
LstSetSelection. . . . .	341
LstSetTopItem . . . . .	341
Menu Functions . . . . .	342
MenuDispose . . . . .	342
MenuDrawMenu. . . . .	343
MenuEraseStatus. . . . .	344
MenuGetActiveMenu. . . . .	345
MenuHandleEvent . . . . .	346
MenuInit . . . . .	347
MenuSetActiveMenu . . . . .	347
Miscellaneous User Interface Functions . . . . .	348
AbtShowAbout . . . . .	348
DayHandleEvent. . . . .	348
KeySetMask . . . . .	349
LocGetNumberSeparators . . . . .	349
Pen Manager Functions . . . . .	350
PenCalibrate. . . . .	350
PenResetCalibration . . . . .	351
Functions for System Use Only. . . . .	351
Progress Manager Functions . . . . .	353
<u>PrgHandleEvent</u>	<u>353</u>
<u>PrgStartDialog</u>	<u>354</u>
<u>PrgStopDialog</u>	<u>355</u>
<u>PrgUpdateDialog</u>	<u>356</u>
<u>PrgUserCancel</u>	<u>357</u>
Rectangle Functions. . . . .	358
RctCopyRectangle . . . . .	358

## Table of Contents

---

RctGetIntersection . . . . .	358
RctInsetRectangle . . . . .	359
RctOffsetRectangle . . . . .	360
RctPtInRectangle . . . . .	361
RctSetRectangle . . . . .	361
Scrollbar Functions . . . . .	362
ScIDrawScrollBar . . . . .	362
ScIGetScrollBar . . . . .	362
ScIHandleEvent . . . . .	363
ScISetScrollBar . . . . .	364
Functions for System Use Only . . . . .	365
Time Selection Functions . . . . .	367
SelectDay . . . . .	367
SelectDayV10 . . . . .	367
SelectTime. . . . .	368
Table Functions. . . . .	369
TblDrawTable . . . . .	369
TblEditing. . . . .	369
TblEraseTable . . . . .	370
TblFindRowData. . . . .	370
TblFindRowID. . . . .	371
TblGetBounds . . . . .	371
TblGetColumnSpacing . . . . .	372
TblGetColumnWidth . . . . .	372
TblGetCurrentField. . . . .	373
TblGetItemBounds . . . . .	373
<u>TblGetItemFont</u> . . . . .	<u>374</u>
TblGetItemInt . . . . .	374
TblGetLastUsableRow . . . . .	375
TblGetNumberOfRows . . . . .	375
TblGetRowData . . . . .	375
TblGetRowHeight . . . . .	376
TblGetRowID . . . . .	376
TblGetSelection . . . . .	377
TblGrabFocus . . . . .	377
TblHandleEvent . . . . .	378

## Table of Contents

---

TblHasScrollBar . . . . .	378
TblInsertRow . . . . .	379
TblMarkRowInvalid . . . . .	379
TblMarkTableInvalid . . . . .	380
TblRedrawTable . . . . .	380
TblReleaseFocus . . . . .	381
TblRemoveRow . . . . .	381
TblRowInvalid. . . . .	382
TblRowSelectable . . . . .	382
TblRowUsable . . . . .	383
TblSelectItem . . . . .	383
TblSetBounds . . . . .	384
TblSetColumnEditIndicator . . . . .	384
TblSetColumnSpacing . . . . .	385
TblSetColumnUsable . . . . .	385
TblSetColumnWidth . . . . .	386
TblSetCustomDrawProcedure . . . . .	387
<u>TblSetItemFont</u> . . . . .	<b>388</b>
TblSetItemInt . . . . .	388
TblSetItemPtr . . . . .	389
TblSetItemStyle . . . . .	390
TblSetLoadDataProcedure. . . . .	391
TblSetRowData . . . . .	392
TblSetRowHeight . . . . .	392
TblSetRowID . . . . .	393
TblSetRowSelectable . . . . .	393
TblSetRowStaticHeight . . . . .	394
TblSetRowUsable . . . . .	394
TblSetSaveDataProcedure . . . . .	395
TblUnhighlightSelection . . . . .	395
Window Functions . . . . .	396
<u>ScrDisplayMode</u> . . . . .	<b>396</b>
WinAddWindow. . . . .	400
WinClipRectangle . . . . .	400
WinCopyRectangle . . . . .	401
WinCreateOffscreenWindow. . . . .	402

## Table of Contents

---

WinCreateWindow . . . . .	403
WinDeleteWindow . . . . .	404
WinDisableWindow . . . . .	404
WinDisplayToWindowPt . . . . .	405
WinDrawBitmap . . . . .	405
WinDrawChars . . . . .	406
WinDrawGrayLine . . . . .	406
WinDrawGrayRectangleFrame. . . . .	407
WinDrawInvertedChars. . . . .	407
WinDrawLine . . . . .	408
WinDrawRectangle. . . . .	408
WinDrawRectangleFrame . . . . .	409
WinDrawWindowFrame . . . . .	409
WinEnableWindow. . . . .	410
WinEraseChars . . . . .	410
WinEraseLine . . . . .	411
WinEraseRectangle . . . . .	411
WinEraseRectangleFrame . . . . .	412
WinEraseWindow . . . . .	412
WinFillLine . . . . .	413
WinFillRectangle . . . . .	413
WinGetActiveWindow . . . . .	414
WinGetClip . . . . .	414
WinGetDisplayExtent. . . . .	415
WinGetDisplayWindow. . . . .	415
WinGetDrawWindow. . . . .	416
WinGetFirstWindow . . . . .	416
WinGetFramesRectangle . . . . .	417
WinGetPattern. . . . .	418
WinGetWindowBounds. . . . .	418
WinGetWindowExtent . . . . .	419
WinGetWindowFrameRect . . . . .	419
WinGetWindowPointer . . . . .	420
WinInitializeWindow . . . . .	420
WinInvertChars . . . . .	421
WinInvertLine . . . . .	421

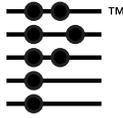
## Table of Contents

---

WinInvertRectangle . . . . .	422
WinInvertRectangleFrame . . . . .	422
WinModal . . . . .	423
WinRemoveWindow . . . . .	423
WinResetClip . . . . .	424
WinRestoreBits . . . . .	424
WinSaveBits . . . . .	425
WinScrollRectangle . . . . .	426
WinSetActiveWindow . . . . .	427
WinSetClip . . . . .	427
WinSetDrawWindow . . . . .	428
WinSetPattern . . . . .	428
WinSetUnderlineMode . . . . .	429
WinSetWindowBounds . . . . .	429
<u>WinValidateHandle</u> . . . . .	<u>430</u>
WinWindowToDisplayPt . . . . .	430
<b>Index . . . . .</b>	<b>431</b>

## Table of Contents

---



# About This Document

---

Developing Palm OS 3.0 Applications, Part I, is part of the Palm OS Software Development Kit. This introduction provides an overview SDK documentation, discusses what materials are included in this document and what conventions are used.

## Palm OS SDK Documentation

The following documents are part of the SDK:

Document	Description
Palm OS 3.0 Tutorial	A number of Phases step developers through using the different parts of the system. Example applications for each phase are part of the SDK.
Developing Palm OS 3.0 Applications. Part I: Interface Management	A programmer's guide and reference document that introduces all important parts of developing an application. See <a href="#">What This Guide Contains</a> for details.
Developing Palm OS 3.0 Applications. Part II. System Management	A programmer's guide and reference document for all system managers, such as the string manager or the system event manager.

## About This Document

### What This Guide Contains

---

Document	Description
Developing Palm OS 3.0 Applications, Part III. Memory and Communications Management	Programmer's guide and reference document for: <ul style="list-style-type: none"><li>• Memory management; both the database manager and the memory manager.</li><li>• The Palm OS communications library for serial communication.</li><li>• The Palm OS net library, which provides basic network services.</li><li>• The exchange manager and IR library, which provide infrared communication capabilities.</li></ul>
Palm OS 3.0 Cookbook	Information about using CodeWarrior for Palm OS to create projects and executables. Also provides a variety of design guidelines, including localization design guidelines.

## What This Guide Contains

This section provides an overview of the chapters in this guide.

- [Chapter 1, “Developing Palm OS Applications.”](#) helps you understand the basic principles of application development. The chapter provides information on:
  - steps involved in creating an application
  - internal structure of an application
  - naming conventions
  - basic hardware
  - the different Palm Computing Platform devices
- [Chapter 2, “Application Control Flow.”](#) explains how applications and the system work together using events and launch codes.
  - Events are posted by the system in response to user input and are then either handled by the system itself or by the application.
  - Launch codes are sent to the top level of the event loop. They are usually sent by the system and require an application response. Applications can, however, send launch codes themselves if desired.

- [Chapter 3, “Palm OS User Interface Resources.”](#) provides detailed information about all the UI resources an application can use a templates for its user interface.
- [Chapter 4, “Palm OS User Interface Objects.”](#) discusses the C structures applications can use to manipulate the UI resources discussed in chapter 3. Each object is related with one or more UI resources and has a number of fields and attributes an application program can change.
- [Chapter 5, “Using Palm OS UI Managers.”](#) discusses the functionality of all managers related to the Palm OS user interface. For each manager, the chapter provides a functionality overview, discussion of how to use that functionality, and list of functions.
- [Chapter 6, “Palm OS Events.”](#) provides reference-style information for each event.
- [Chapter 7, “Palm OS User Interface Functions.”](#) provides reference-style information for each UI function.

## Conventions Used in This Guide

This guide uses the following typographical conventions:

<b>This style...</b>	<b>Is used for...</b>
<code>fixed width font</code>	Code elements such as function, structure, field, bitfield.
<u>fixed width underline</u>	Emphasis (for code elements).
<b>bold</b>	Emphasis (for other elements).
<a href="#">blue and underlined</a>	Hot links.
<u>black and underlined</u>	3.0 function names (headings only)
<u>red and underlined</u>	3.0 function names (Table of Contents only)

## What's New in Palm OS 3.0

Version 3.0 of the Palm OS brings changes to many areas of the system. Some programming interfaces have been extended and others are brand new. For those readers familiar with previous versions of the Palm OS API, this section gives a brief overview of what is new, what has changed, and where you can find documentation on the new APIs. Other documentation revisions are also noted.

### General Information

The configuration of system memory includes the following 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.

For more details on memory management under OS 3.0, see the chapter [“Palm OS Memory Management” on page 21](#) of Part III. This chapter now presents an overview of memory management and heap usage for all versions of Palm OS.

There is a new system version number for the Palm III device, as returned by [FtrGet](#). It is 0x03003000.

Palm III devices contain a unique serial number that is accessible to developers and users. For more information, see [“Retrieving the ROM Serial Number” on page 51](#) of Part 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. The Launcher can beam your application to another device or you can suppress beaming. The new Launcher provides a list mode which allows users to see more installed applications at once. For more information, see [“Application Launcher” on page 70](#) of Part II.

## New Launch Codes

Two new launch codes have been added to support the exchange manager:

- [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.

## Dynamic User Interface Objects

Palm OS 3.0 provides functions that can be used to create forms and form elements at run time. Most applications will never need to change any user interface elements at run time—the built-in applications don't do so, and the Palm user interface guidelines discourage it. For more information, see the section [“Dynamic User Interface Objects” on page 175](#) of Part I. The following new functions are included in the dynamic user interface API:

- [CtlNewControl](#)
- [CtlValidatePointer](#)
- [FldNewField](#)
- [FrmNewBitmap](#)
- [FrmNewForm](#)
- [FrmNewGadget](#)
- [FrmNewLabel](#)
- [FrmRemoveObject](#)
- [FrmValidatePtr](#)
- [LstNewList](#)
- [WinValidateHandle](#)

## Font Functions

The Palm OS 3.0 provides a new font (`largeBoldFont`), two new font manipulation routines ([FontSelect](#) and [FntDefineFont](#)), and support for the use of custom fonts. For more information, see the section [“Font Functions” on page 262](#) of Part I.

## About This Document

### *What's New in Palm OS 3.0*

---

## Progress Manager

The progress manager is a new manager that provides support for displaying and updating a progress dialog to the user during lengthy operations (such as communications). For more information, see the section [“The Progress Manager” on page 186](#) of Part I.

New functions include:

- [PrgHandleEvent](#)
- [PrgStartDialog](#)
- [PrgStopDialog](#)
- [PrgUpdateDialog](#)
- [PrgUserCancel](#)

## File Streaming API

The file streaming functions in Palm OS 3.0 let you work with large blocks of data (larger than 64 KB) in a manner similar to traditional desktop file systems. The File Streaming API is derived from the C programming language's `<stdio.h>` interface. For more information, see the section [“File Streaming Application Program Interface” on page 28](#) of Part II. New functions include:

- [FileClearerr](#)
- [FileClose](#)
- [FileControl](#)
- [FileDelete](#)
- [FileDmRead](#)
- [FileEOF](#)
- [FileError](#)
- [FileFlush](#)
- [FileGetLastError](#)
- [FileOpen](#)
- [FileRead](#)
- [FileReadLow](#)
- [FileRewind](#)
- [FileSeek](#)
- [FileTell](#)

- [FileTruncate](#)
- [FileWrite](#)

## Sound Manager

The sound manager in Palm OS 3.0 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 [“The Sound Manager” on page 35](#) of Part II. New functions include:

- [SndCreateMidiList](#)
- [SndPlaySMF](#)
- [SndDoCmd](#) (enhanced in 3.0)

The sound manager section has also been revised to describe how to use sound preferences correctly on various versions of Palm OS.

## Exchange Manager

The exchange manager is a new manager in Palm OS 3.0 that provides a simple interface for Palm OS applications to send and receive typed data from any number of remote devices and protocols. Initially, the exchange manager supports infrared “beaming” of information between Palm devices and other external devices. For more information, see the chapter [“Exchange Manager” on page 279](#) of Part III.

## IR Library

The IR library is a new shared library of functions that provide a direct interface to the infrared (IR) communications capabilities of Palm OS 3.0. It is designed for applications that need more direct access to the IR capabilities than the exchange manager provides. The IR support provided by the Palm OS is compliant with the IrDA specifications. For more information, see the chapter [“IR Library” on page 295](#) of Part III.

## About This Document

### *What's New in Palm OS 3.0*

---

## Miscellaneous New Functions in 3.0

The following other new functions were added in Palm OS 3.0:

[ScrDisplayMode](#)

[SysGetAppInfo](#) (system use only)

[SysGetOSVersionString](#)

[SysGetRomToken](#)

[SysGetStackInfo](#)

[SysGremlins](#)

[TblGetItemFont](#)

[TblSetItemFont](#)

The following functions existed in the system previously, but were not previously documented:

[RctCopyRectangle](#)

[RctGetIntersection](#)

[RctInsetRectangle](#)

[RctOffsetRectangle](#)

[RctPtInRectangle](#)

[RctSetRectangle](#)

The following event type existed in the system previously, but was not previously documented:

[frmGotoEvent](#)

## Existing Functions that Changed in 3.0

Only two functions that existed in 2.0 were changed in 3.0. These are [CategoryEdit](#) and [SysBatteryInfo](#). If you are using these functions in existing applications, you can continue using the old APIs by calling the functions [CategoryEditV20](#) and [SysBatteryInfoV20](#). However, we recommend that you begin using the new changed APIs for future compatibility.

## Documentation Revisions

The documentation for the following events and functions has been revised to correct inaccuracies or add clarification:

[lstEnterEvent](#)

[CategoryCreateList](#)

[DateAdjust](#)

[FrmCustomAlert](#)

[FrmGetControlGroupSelection](#)

[FldGetInsPtPosition](#)

[FldSetSelection](#)

[MenuEraseStatus](#)

[StrNCat](#)

[SysBinarySearch](#)

[SysCreateDataBaseList](#)

[SysInsertionSort](#)

[TblSetLoadDataProcedure](#) (loadDataCallback)

[TblSetSaveDataProcedure](#) (saveDataCallback)

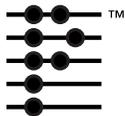
The following other section of the documentation is new:

[“Writing Robust Code” on page 37](#) of Part I.

## **About This Document**

*What's New in Palm OS 3.0*

---



# Developing Palm OS Applications

---

This chapter helps you understand the basic principles of Palm OS application development. It discusses these topics:

- [Overview of Application Development](#) explains the steps involved in creating an application.
- [Internal Structure of an Application](#) provides some information about resources the system creates for each application and how they are used.
- [Naming Conventions](#) briefly explains naming conventions used for functions and structures.
- [Basic Hardware](#) gives some background information about the Palm Computing Platform devices.
- [Different Palm Computing Platform Devices](#) discusses how to make 1.0 applications run on the new devices, and how to run 3.0 applications on older devices.

## Overview of Application Development

This section provides an overview of the application development process for Palm Computing Platform devices. It introduces the different components of an application in the order that you'll most likely work with them, and provides many links to related sections in this guide and pointers to other relevant documentation included in your developer package.

You learn about these topics:

- [Designing UI and Program Functionality](#)
- [Constructing UI Resources](#)
- [Using Managers and Filling Out the Program Logic](#)

- [Writing Robust Code](#)
- [Building, Debugging, and Testing](#)
- [Building the Application and Running it on the Palm Device](#)
- [Using Other Components of the SDK](#)

## **Designing UI and Program Functionality**

The first step in application development is to envision what users will do as they interact with your application. After that, it's useful to implement a small prototype and have some users interact with it. When you're satisfied with the basic interface and user interaction, you can move on from the prototype to a complete application.

This section looks at the steps involved in creating a working user interface.

### **Designing Screen Layout and User Interaction**

Careful UI design is critical for a Palm OS application because using a Palm Computing Platform device differs from using other computers. Here are a few points to consider when designing your application:

#### **Do this...**

Pay attention to pen-based user input paradigms.

Plan integration with the desktop early.

Offload some computationally intensive tasks to the desktop; use the device as a satellite viewer.

#### **Because of this...**

Pen-based user input differs from keyboard-based user input.

Your conduit/backup strategy and your integration with desktop software can make your program much more useful.

Device runs on batteries and doesn't have the same processing power than the desktop PC.

**Do this...**

Limit data input where possible.

Design the layout carefully.  
Strive for a balance between providing enough information and overcrowding the screen.

**Because of this...**

Graffiti and the popup keyboard are useful tools, but not as easy to use as a regular keyboard.

Screen size is very limited.

The Palm OS development team has put together a set of design guidelines that were used as the basis for the four applications resident on the device (MemoPad, Address Book, etc.). These guidelines are summarized in Chapter 1 of the “Palm OS Cookbook.” Some information, such as recommended font size or border width, is included for each resource in [Chapter 3, “Palm OS User Interface Resources.”](#)

---

**Note:** Follow the design guidelines in chapter 1 of the “Palm OS Cookbook” to make your application easier to learn and to use.

---

## Constructing UI Resources

The resource templates that were used to implement all the applications resident on the device are provided with your development environment.

The Palm OS 3.0 SDK Constructor tool lets you use the resource templates to create your own buttons, popup lists, menus, and other parts of the user interface.

The process of creating new resources is described in detail in the tutorial; the basic process consists of entering values into the attribute fields of the resource templates. Each resource has to have an ID; and may also need a width, height, label, or other attributes. The Constructor tool assigns the ID.

Macintosh users can also use ResEdit to create resources, but have to assign the IDs explicitly in that case.

The recommended (or required) values for the different fields in each resource are provided in [Chapter 3, “Palm OS User Interface Resources.”](#) The “Palm OS Tutorial” provides “recipes” for creating each resource type in the Tutorial’s “Resource Recipes” chapter.

When you build your program, the system converts the UI resources into data structures that the system can work with. Different resource types map to different data structures, that is, UI object type. For example, menu resources map to C structures that have system-defined behavior for turning highlighting on and off. Fields have system-defined behavior for positioning input cursors and processing user input. Note that there isn’t a 1:1 mapping between resources and UI objects. This is explained in more detail in the relevant chapters.

The operating system provides quite a bit of default functionality for each UI object type. Your program logic can use, replace, or extend that functionality. Detailed information on all structures and their fields is provided in [Chapter 4, “Palm OS User Interface Objects.”](#)

## Using Managers and Filling Out the Program Logic

To successfully build a Palm OS application, you have to understand how the system itself is structured and how to structure your application. You learn about [Using Events and Launch Codes](#) and [Using Palm OS Managers](#).

### Using Events and Launch Codes

Palm OS applications are single-threaded, event-driven programs. The events are generated by the system, based on user input and system interrupts. The program logic may generate events as well. The programs are structured as a series of event handlers dispatched from a single event loop in each program.

Launch codes allow the system (or another application) to send a request to an application at the top level. For example, one application that supports global find may bring up the Find dialog and the

system will query all currently loaded applications that handle the global find. In response to a launch code, an application doesn't necessarily display its user interface; instead, it only performs the requested action. This is described in more detail in [How Launch Codes Control an Application](#).

---

**Note:** To make your application interact appropriately with other applications on the device and to avoid problems later, read Chapter 1 of the Palm OS Cookbook.

---

## Using Palm OS Managers

The Palm OS system API is divided into functional areas called managers. Each manager has a distinct three-letter prefix used on all API calls and structures and is discussed separately below.

- All UI related managers, such as the graffiti or key manages, are discussed in [“Using Palm OS UI Managers.”](#)
- All system related managers, such the string or system event manager, are discussed in [“Using Palm OS System Managers”](#) of “Developing Palm OS 3.0 Applications, Part II.”
- The memory manager, data manager, and resource manager are explained in [“Palm OS Memory Management”](#) of “Developing Palm OS 3.0 Applications, Part III.”
- The serial communications API is explained in [“Palm OS Communications”](#) of “Developing Palm OS 3.0 Applications, Part III.”
- The exchange manager API is explained in [“Exchange Manager”](#) of “Developing Palm OS 3.0 Applications, Part III.”

## Writing Robust Code

To make your programs more robust and to increase their compatibility with the next generation of Palm Computing products, it is strongly recommended that you follow the guidelines and practices outlined in this section.

### Check Assumptions

You can write defensive code by adding frequent calls to the [ErrNonFatalDisplayIf](#) function, which enables your debug

builds to check assumptions. Many bugs are caught in this way, and these “extra” calls don't weigh down your shipping application. You can keep more important checks in the release builds by using the [ErrFatalDisplayIf](#) function.

#### **Avoid reading and writing to NULL (or low memory)**

When calling functions that allocate memory ([MemSet](#), [MemMove](#) and similar functions) make sure that the pointers they return are non-NULL. (If you can do better validation than that, so much the better.) Also check that pointers your code obtains from structures or other function calls are not NULL. Consider adding to your debug build a `#define` that overrides [MemMove](#) (and similar functions) with a version that validates the arguments passed to it.

#### **Use Dynamic Heap Space Frugally**

It is important not to use the extra dynamic heap space available on Palm OS 3.0 units unless it is truly necessary to do so. Wasteful use of heap space may limit your application to running only on devices having 2MB of memory—which prevents it from running on the very large number of units already in the marketplace.

Note that some system services, such as the IrDA stack or the Find window, can require additional memory while your application is running; for example, if the unit starts to receive a beam or other external input, the system may need to allocate additional heap space for the incoming data. Don't use all available dynamic memory just because it's there; instead, consider using the storage heap for working with large amounts of temporary data.

#### **Check Result Codes When Allocating Memory**

Because future devices may have larger or smaller amounts of available memory, it is always a good idea to check result codes carefully when allocating memory. It's also good practice to use the storage heap (and possibly file streams) to work with large objects.

#### **Avoid allocating zero-length objects**

It's not valid to allocate a zero-byte buffer, or to resize a buffer to zero bytes. Palm OS 2.0 and previous releases allowed this practice, but future revisions of the OS may not permit zero-length objects.

### **Avoid making assumptions about the screen**

The location of the screen buffer, its size, and the number of pixels per bit aren't set in stone—they might well change. Don't hack around the windowing and drawing functions. If you are going to hack the hardware to circumvent the APIs, save the state and return the system to that saved state when you quit.

### **Don't access globals or hardware directly**

Global variables and their locations can change; to avoid mishap, use the documented API functions and disable your application if it is run on anything but a tested version of the OS. Future devices might run on a different processor than the current one.

Similarly, don't hardcode references to cards. Although current Palm OS hardware provides only a single card slot, this may not always be the case. Thus, when calling functions that manipulate cards, such as database manager and file streaming functions, pass a variable that references the target card, rather than passing a hard-coded reference to card 0.

### **Don't overfill the stack**

Allocating large numbers of local variables (or extremely large local variables) can result in hard-to-debug heap corruption. The stack is only about 4k; be stingy with stack-based variables!

### **Built-in apps can change**

The format and size of the preferences (and data) for the built-in applications is subject to change. Write your code defensively, and consider disabling your application if it is run on an untested version of the OS.

### **Don't use desktop C libraries on Palm OS**

Avoid using functions from standard desktop C libraries, as they tend to slow performance and enlarge programs significantly. Many will not work at all on Palm devices. Use functions provided by the Palm OS managers instead.

## **Building, Debugging, and Testing**

To build your application for initial debugging and testing, you use the CodeWarrior Interactive Development Environment (IDE) to build an executable. Documentation for the CodeWarrior IDE is provided with CodeWarrior.

On the Macintosh, you can then use the Palm Simulator to run the executable on a simulated Palm Computing Platform device on the Macintosh screen. You can interact with the simulated buttons, menus, or fields, and even enter Graffiti characters using the mouse. You can also use the Simulator to test your application using an automated test suite called Gremlins. Using the Simulator is discussed in detail in Chapter 2, “Using the Palm OS Simulator,” of the “Palm OS Cookbook.”

On both Macintosh and PC, you can use the CodeWarrior Debugger to download the application to a device connected with the Desktop computer and test it there. The CodeWarrior debugging environment is also documented in the CodeWarrior Documentation folder.

## **Building the Application and Running it on the Palm Device**

When you’ve completed building and testing the application with the Simulator, you can build a second project inside CodeWarrior that lets you run your application on the device.

This process is described in Chapter 3, “Debugging in Standalone Mode on the Device” of the Palm OS Cookbook and in Phase 20 of the tutorial.

Phase 20 of the Palm OS Tutorial provides a sample project and step-by-step instructions for setting up a project to build an executable and for downloading and running the application on the device.

---

**Note:** When using the Palm OS 1.0 SDK, developers had to create a Makefile to create an executable they could download and run on the device. This is no longer necessary; instead, developers use a project that's built with different settings.

---

## Using Other Components of the SDK

The Palm Computing Platform has provided the following additional items in the development kit to help you come up to speed quickly:

- The Palm OS tutorial provides step-by-step examples of developing an application from start to finish in its more than twenty phases. Examples, both resources and code that is incrementally changed, are included.
- The actual source code for the four PIM applications on the Palm device is included as examples on your SDK CD. The code can be a valuable aid when you develop your own program. The software development kit provides a royalty-free license that permits you to use any or all of the source code from the examples in your application.
- The Palm OS net library provides basic networking capabilities, compatible with the Berkeley Sockets API. The net library is discussed in “Developing Palm OS 3.0 Applications, Part III.”

## Internal Structure of an Application

Every application running under Palm OS must have certain minimum system (not UI) resources defined to be recognized by the Palm OS system software. These required resources are created for your application by the development environment. You may find that you need additional, application-specific resources.

Resources consist of a type and an ID, where the type is a 4-byte ASCII string like ‘code’ and the ID is a decimal integer preceded by a pound sign.

## The 'code' #1 Resource

The system creates a 'code' #1 resource for every application. This resource is the entry point for the application and is where application initialization is performed. When the Palm OS launches an application, it starts executing at the first byte of the 'code' #1 resource. All of the application code that you provide is included in this resource as well.

Typically, some startup code provided with the Palm Computing Platform development environment is linked in with your application code. This startup code works as follows:

- The startup code performs application setup and initialization.
- The startup code calls your `main` routine.
- When your `main` routine exits, control is returned to the startup code, which performs any necessary cleanup of your application and returns control to the Palm OS system software.

## The 'pref' #0 Resource

The system creates a 'pref' #0 resource for every application. This resource contains startup information necessary for launching your application. The resource includes

- Required stack size
- Dynamic heap space required
- Task priority

Note that although the 'pref' #0 resource must be present, it's mainly for future use because user-interface applications currently don't get their own stack or priority.

## The 'code' #0 and 'data' #0 Resources

The 'code' #0 and 'data' #0 resources contain the required size of your global data and an image of the initialized area of that global data. When your application is launched, the system allocates a memory chunk in the dynamic heap that's big enough to hold all your globals. The 'data' #0 resource is then used to initialize those globals.

## Naming Conventions

The following conventions are used throughout the Palm OS API:

- Functions start with a capital letter.
- All functions belonging to a particular manager start with a two- or three-letter prefix, such as "Ctl" for control functions or "Ftr" for functions that are part of the feature manager.
- Events and other constants start with a lowercase letter.
- Structure elements start with a lowercase letter.
- Global variables start with a capital letter.
- Typedefs start with a capital letter and end with "type" (for example, `DateFormatType`, found in `DateTime.h`).
- Macintosh ResEdit resource types usually start with a lowercase letter followed by three capital letters, for example `tSTR` or `tTBL`. (Customized Macintosh resources provided with your developer package are all uppercase, for example, `MENU`. Some resources, such as `Talt`, don't follow the conventions.)
- Members of an enumerated type start with a lowercase prefix followed by a name starting with a capital letter, as follows:

```
enum formObjects {  
    frmFieldObj,  
    frmControlObj,  
    frmListObj,  
    frmTableObj,  
    frmBitmapObj,  
    frmLineObj,  
    frmFrameObj,  
};
```

```
    frmRectangleObj,  
    frmLabelObj,  
    frmTitleObj,  
    frmPopupObj,  
    frmGraffitiStateObj,  
    frmGadgetObj};  
typedef enum formObjects FormObjectKind;
```

## Basic Hardware

This section helps you understand the device for which you're developing your application.

It discusses [RAM and ROM](#), [Modes of Operation](#), [Palm OS Connectivity](#), [Real-Time Clock and Timer](#), [Palm OS Device Screen and Sound Generation](#), and [Palm OS Device Reset Switch](#).

### RAM and ROM

The 1.0, 2.0, and 3.0 versions of the Palm OS run on the Motorola 68328 "DragonBall" processor. The first memory card shipped with the device has 128K of pseudostatic RAM and 512K of ROM for the system software and application code. A portion of the RAM (32K) is reserved for system use and is not available for storing user data. Both the ROM and RAM are on a memory module that users can replace. The Palm Computing Platform device does not have a disk drive or PCMCIA support.

---

**Note:** The PalmPilot Professional has additional RAM for use by the network library. This memory is not for application use.

---

### Modes of Operation

To minimize power consumption, the operating system dynamically switches between three different modes of operation: sleep mode, doze mode, and running mode.

- In **sleep mode**, the device looks like it's turned off: the display is blank, the digitizer is inactive, and the main clock is stopped. The only circuits still active are the real-time clock and interrupt generation circuitry.

The device enters this mode when there is no user activity for a number of minutes or when the user presses the off button. The device comes out of sleep mode only when there is an interrupt, for example, when the user presses a button.

- In **doze mode**, the main clock is running, the device appears to be turned on, and the processor's clock is running but it's not executing instructions (that is, it's halted). When the processor receives an interrupt, it comes out of halt and starts processing the interrupt.

The device enters this mode whenever it's on but has no user input to process.

- In **running mode**, the processor is actually executing instructions.

The device enters this mode when it detects user input (like a tap on the screen) while in doze mode or when it detects an interrupt while in doze or sleep mode. The device stays in running mode only as long as it takes to process the user input (most likely less than a second), then it immediately re-enters doze mode.

To maximize battery life, the processor on the Palm Computing Platform device is kept out of running mode as much as possible. Any interrupt generated on the device must therefore be capable of "waking" up the processor. The processor can receive interrupts from the serial port, the hard buttons on the case, the button on the cradle, the programmable timer, the memory module slot, the real-time clock (for alarms), the low-battery detector, and any built-in peripherals such as a pager or modem.

## Palm OS Connectivity

The Palm Computing Platform device uses its serial port for implementing desktop PC connectivity or other external communication. The serial communication is fully interrupt-driven for receiving data. Currently, interrupt-driven transmission of data is not

implemented in software, but the hardware does support it. Five external signals are used for this communication:

- SG (signal ground)
- TxD (transmit data)
- RxD (receive data)
- CTS (clear to send)
- RTS (request to send)

The Palm Computing Platform device has an external connector that provides:

- Five serial communication signals
- General-purpose output
- General-purpose input
- Cradle button input

## Real-Time Clock and Timer

The Palm Computing Platform device has a real-time clock and programmable timer as part of the 68328 processor. The real-time clock maintains the current time even when the system is in sleep mode (turned off). It's capable of generating an interrupt to wake the device when an alarm is set by the user. The programmable timer is used to generate the system tick count interrupts (100 times/second) while the processor is in doze or running mode. The system tick interrupts are required for periodic activity such as polling the digitizer for user input, key debouncing, etc.

The Palm Computing Platform device has one memory module socket for installing modules that may contain ROM or RAM storage.

## Palm OS Device Screen and Sound Generation

The first version of the Palm Computing Platform device has an LCD screen of 160x160 pixels. The LCD controller built into the 68328 maps a portion of system memory to the LCD. Currently, the

software only supports 1 bit/pixel monochrome graphics, although the controller can support 2 bits/pixel gray scale.

The Palm Computing Platform device has a built-in digitizer overlaid onto the LCD screen and extending about an inch below the screen. This digitizer is capable of sampling accurately to within 0.35 mm (.0138 in) with up to 50 accurate points/second. When the device is in doze mode, an interrupt is generated when the pen is first brought down on the screen. After a pen down is detected, the system software polls the pen location periodically (every 20 ms) until the pen is again raised.

The Palm Computing Platform device has primitive sound generation. A square wave is generated directly from the 68328's PWM circuitry. There is frequency, duration, and volume control. Additionally, OS 3.0 supports creating and playing standard MIDI sounds.

## **Palm OS Device Reset Switch**

Any reset is normally performed by sticking a bent-open paper clip or a large embroidery needle into the small hole in the back of the device. This hole, known as the “reset switch” is above and to the right of the serial number sticker (on Palm III devices). Depending on additional keys held down, the reset behavior varies, as follows:

### **Soft Reset**

A soft reset clears all of the dynamic heap (Heap 0, Card 0). The storage heaps remain untouched. The operating system restarts from scratch with a new stack, new global variables, restarted drivers, and a reset comm port. All applications on the device receive a `SysAppLaunchCmdReset` message.

### **Soft Reset + Up Arrow**

Holding the up-arrow down while pressing the reset switch with a paper clip causes the same soft reset logic with the following two exceptions:

- The `SysAppLaunchCmdReset` message is not sent to applications. This is useful if there is an application on the device that

## Developing Palm OS Applications

### *Different Palm Computing Platform Devices*

---

crashes upon receiving this message (not uncommon) and therefore prevents the system from booting.

- The OS won't load any system patches during startup. This is useful if you have to delete or replace a system patch database. If the system patches are loaded and therefore open, they cannot be replaced or deleted from the system.

### **Hard Reset**

A hard reset is performed by pressing the reset switch with a paper clip while holding down the power key. This has all the effects of the soft reset. In addition, the storage heaps are erased. As a result, all programs, data, patches, user information, etc. are lost. A confirmation message is displayed asking the user to confirm the deletion of all data.

The `SysAppLaunchCmdReset` message is sent to the applications at this time. If the user selected the "Delete all data" option, the digitizer calibration screen comes up first. The default databases for the four main applications is copied out of the ROM.

If you hold down the up arrow key when the "Delete all data" message is displayed, and then press the other four application buttons while still holding the up arrow key, the system is booted without reading the default databases for the four main applications out of ROM.

## Different Palm Computing Platform Devices

In spring 1998, a new Palm Computing Platform device became available. As a result, there are now 4 devices:

- Palm Computing Platform 1.0 device (Pilot 1000 and Pilot 5000)
- Palm Computing Platform 2.0 device (PalmPilot and PalmPilot Professional)
- Palm Computing Platform 3.0 device (Palm III)

This section summarizes migrating to Palm Computing Platform 3.0 by discussing [Running Older Applications on the 3.0 Device](#),

[Compiling Older Applications With SDK Version 3.0](#), [Using OS Version 3.0 Features](#), and [Retrieving the System Version Number](#).

---

**Caution:** The sample PIM applications (Date Book, Address Book, Memo Pad, To Do List) do not have OS version checking code in them because they are normally built into a 3.0 ROM and the check is unnecessary.

Compiling these samples and running them on an older device will cause the device to crash, but will not cause the loss of any data.

---

## Running Older Applications on the 3.0 Device

As a rule, all Palm OS applications developed with the Palm Computing Platform 1.0 or 2.0 SDK should run error-free on a 3.0 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

## Compiling Older Applications With SDK Version 3.0

If you want to compile your older application under version 3.0, you need to be aware of a number of functions with a changed API. For any of these functions, the old function still exists with a V10 (“v one zero”) or a V20 suffix.

## Developing Palm OS Applications

### *Different Palm Computing Platform Devices*

---

You can choose one of two options:

- Change the function name to keep using the old API. Your 1.0 application will then run error free on a 3.0 device.
- Update your application to use the new API. The application will then run error free and have access to some new functionality.

### **Using OS Version 3.0 Features**

Because Palm OS applications can run on different operating system versions (on the different devices), it's important your application checks that the functionality it uses is actually supported. Most notably:

- All applications that use 3.0 features need to run on a Palm Computing Platform device version 3.0 (Palm III). They won't run on Palm Computing Platform 1.0 or 2.0 devices.
- All applications that use 2.0 features need to run on a Palm Computing Platform device version 2.0 or 3.0 (PalmPilot, PalmPilot Professional, or Palm III). They won't run on a Palm OS 1.0 device.
- All applications that use the network library can run only on Palm III or PalmPilot Professional systems.

### **Running 3.0 Applications on an Older Device**

If you're writing an application that doesn't use any of the new features in 3.0, that application can run on a 2.0 device without any further modification. If it also doesn't use any 2.0-specific features, then it can also run on a 1.0 device without further modification.

3.0 applications are fully data-compatible with 1.0 and 2.0 applications.

## Retrieving the System Version Number

To retrieve the system version number, call:

```
FtrGet(sysFtrCreator, sysFtrNumROMVersion);
```

The system returns:

- 0x01003001 for the 1.0 device
- 0x02003000 for both PalmPilot 2.0 and PalmPilot Professional
- 0x03003000 for the Palm III device

A more detailed discussion of version checking is in Chapter 1 of the Palm OS Cookbook.

## Retrieving the ROM Serial Number

Each flash ROM-based Palm III device holds a 12-digit serial number that identifies it uniquely. (Earlier devices do not have this identifier.) The serial number is held in a displayable text buffer with no null terminator. The user can view the serial number in the [Application Launcher](#) application. (The popup version of the Launcher does not display the serial number.) The Application Launcher on Palm III devices also displays to the user a checksum digit that you can use to validate user entry of the serial number.

To retrieve the ROM serial number programmatically, pass the `sysROMTokenSnum` selector to the [SysGetRomToken](#) function. If the [SysGetRomToken](#) 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.

The `DrawSerialNumOrMessage` function shown in [Listing 1.1](#) retrieves the ROM serial number, calculates the checksum, and draws both on the screen at a specified location. If the device has no serial number, this function draws a message you specify. This function accepts as its input a pair of coordinates at which it draws output, and a pointer to the message it draws when a serial number is not available.

**Listing 1.1 DrawSerialNumOrMessage**

---

```
static void DrawSerialNumOrMessage(int x, int y,
                                   CharPtr noNumberMessage)
{
    CharPtr bufP;
    Word    bufLen;
    Word    retval;
    Short   count;
    Byte    checksum;
    char    checksumStr[2];
    // holds the dash and the checksum digit

    retval = SysGetROMToken (0, sysROMTokenSnum,
                            (BytePtr*) &bufP, &bufLen);
    if ((!retval) && (bufP) && ((Byte) *bufP != 0xFF)) {
        // there's a valid serial number!
        // Calculate the checksum: Start with zero, add each digit,
        // then rotate the result one bit to the left and repeat.
        checksum = 0;
        for (count=0; count<bufLen; count++) {
            checksum += bufP[count];
            checksum = (checksum<<1) | ((checksum & 0x80) >> 7);
        }
        // Add the two hex digits (nybbles) together, +2
        // (range: 2 - 31 ==> 2-9, A-W)
        // By adding 2 to the result before converting to ascii,
        // we eliminate the numbers 0 and 1, which can be
        // difficult to distinguish from the letters O and I.
        checksum = ((checksum>>4) & 0x0F) + (checksum & 0x0F) + 2;

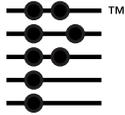
        // draw the serial number and find out how wide it was
        WinDrawChars(bufP, bufLen, x, y);
        x += FntCharsWidth(bufP, bufLen);

        // draw the dash and the checksum digit right after it
        checksumStr[0] = '-';
        checksumStr[1] =
            ((checksum < 10) ? (checksum +'0'):(checksum -10 +'A'));
    }
}
```

```
WinDrawChars (checksumStr, 2, x, y);  
}  
else // there's no serial number  
// draw a status message if the caller provided one  
if (noNumberMessage)  
    WinDrawChars(noNumberMessage, StrLen(noNumberMessage), x, y);  
}
```

---





# Application Control Flow

---

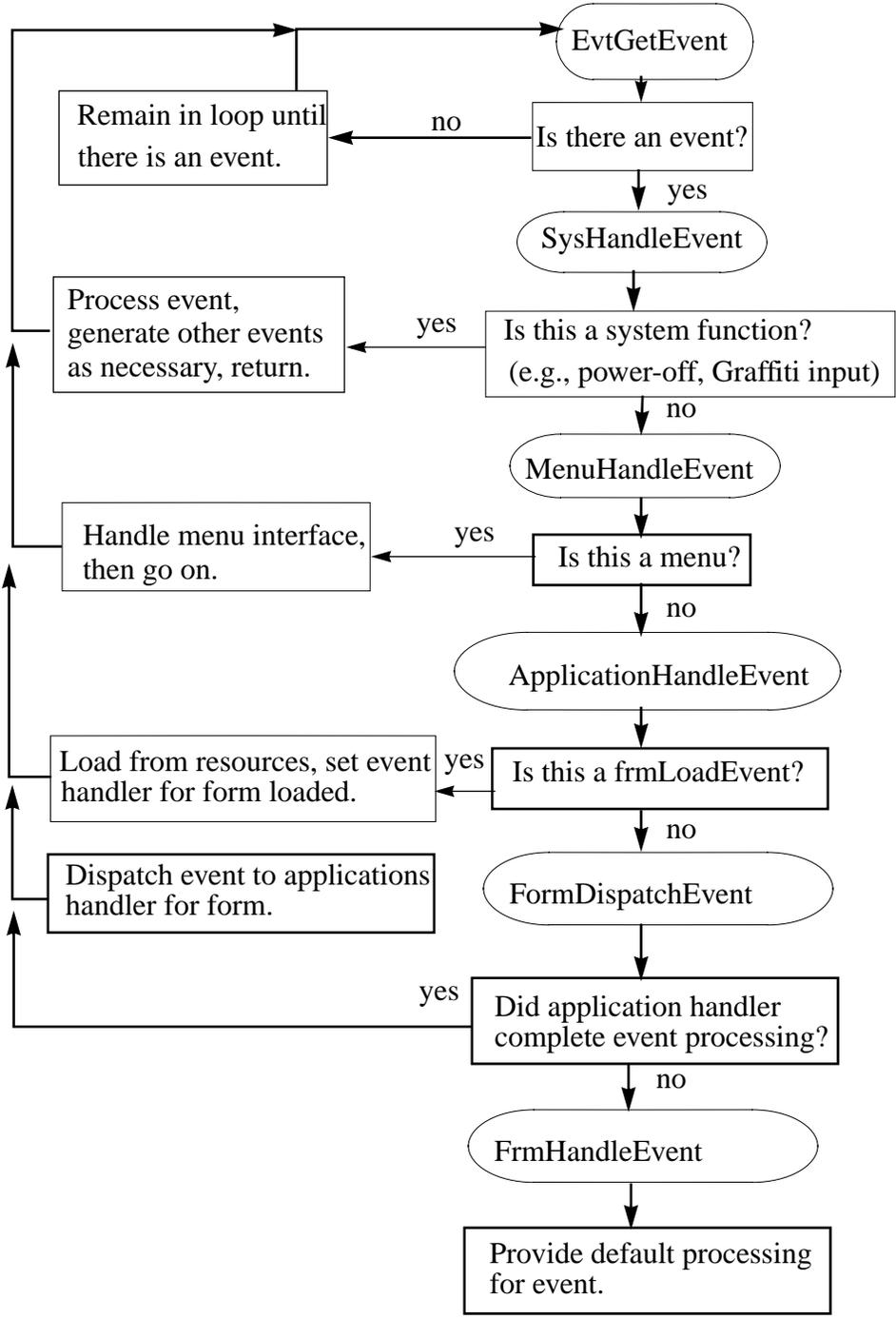
Palm OS applications are generally single-threaded, event-driven programs. They may use predefined UI elements (sometimes referred to as UI objects) or they may create their own UI elements. All applications must use the memory and data management facilities provided by the system and must be considerate of the system and other applications by periodically allowing system event handlers access to the event flow.

The flow of control in Palm OS is driven by two different mechanisms, discussed in some detail in this chapter:

- [How Events Control an Application](#) discusses the event manager, the main interface between the Palm OS system software and an application. It discusses in some detail what an application does in response to user input, providing code fragments as examples where needed.
- [How Launch Codes Control an Application](#) discusses how an application handles requests for immediate action at its top level (`PilotMain`). For example, there are launch codes for launching an application, for telling an application to search its data for a text string, and for notifying an application that data has been synchronized. Using launch codes, an application can request information or actions from another application.

[Figure 2.1](#) illustrates control flow in a typical application.

**Figure 2.1 Control Flow in a Typical Application**



## How Events Control an Application

This section starts with a high-level overview of the stages of a Palm OS application, then provides more information on the event loop.

Note that each event is discussed in some detail in [Chapter 6, “Palm OS Events.”](#) The event flow for each User Interface resource is discussed in [Chapter 3, “Palm OS User Interface Resources.”](#) The event flow for each User Interface object is discussed in [Chapter 4, “Palm OS User Interface Objects.”](#)

### Basic Application Stages

When an application receives the launch code `sysAppLaunchCommandNormalLaunch` (see [How Launch Codes Control an Application](#)), it begins with a startup routine, then goes into an event loop, and finally exits with a stop routine.

- [The Startup Routine](#) is the application’s opportunity to perform actions that need to happen once, and only once, at startup. A typical startup routine opens databases, reads saved state information (such as UI preferences), and initializes the application’s global data.
- [The Event Loop](#) fetches events from the queue and dispatches them, taking advantage of default system functionality as appropriate.
- [The Stop Routine](#) provides an opportunity for the application to perform cleanup activities before exiting. Typical activities include closing databases and saving state information.

The following sections look at each of the stages in some detail. Note that for each phase, Palm OS provides a default behavior that can help you keep application code to a minimum. If your application has special requirements, your application may instead handle the bulk of the work itself.

## The Startup Routine

During the startup routine, your application should perform these actions:

1. Get system-wide preferences (for example for numeric or date and time formats) and use them to initialize global variables that will be referenced throughout the application.
2. Find the application database by creator type. If none exists, create it and initialize it.
3. Get application-specific preferences and initialize related global variables.
4. Initialize any other global variables.

[Listing 2.1](#) shows an example `StartApplication` function from the datebook application.

### Listing 2.1 StartApplication from Datebook.c

---

```
static Word StartApplication (void)
{
    Word error = 0;
    Err err = 0;
    UInt mode;
    DateTimeType dateTime;
    DatebookPreferenceType prefs;
    SystemPreferencesType sysPrefs;
    Word prefsSize;

    // Determine if secret records should be displayed.
    PrefGetPreferences (&sysPrefs);
    HideSecretRecords = sysPrefs.hideSecretRecords;

    if (HideSecretRecords)
        mode = dmModeReadWrite;
    else
        mode = dmModeReadWrite | dmModeShowSecret;

    // Get the time formats from the system preferences.
    TimeFormat = sysPrefs.timeFormat;
```

```
// Get the date formats from the system preferences.
LongDateFormat = sysPrefs.longDateFormat;
ShortDateFormat = sysPrefs.dateFormat;

// Get the starting day of the week from the system preferences.
StartDayOfWeek = sysPrefs.weekStartDay;

// Get today's date.
TimSecondsToDateTime (TimGetSeconds (), &dateTime);
Date.year = dateTime.year - firstYear;
Date.month = dateTime.month;
Date.day = dateTime.day;

// Find the application's data file.  If it doesn't exist
// create it.
ApptDB = DmOpenDatabaseByTypeCreator(datebookDBType,
                                     sysFileCDatebook, mode);
if (! ApptDB)
{
    error = DmCreateDatabase (0, datebookDBName, sysFileCDatebook,
                             datebookDBType, false);
    if (error) return error;

    ApptDB = DmOpenDatabaseByTypeCreator(datebookDBType,
                                         sysFileCDatebook, mode);
    if (! ApptDB) return (1);

    error = ApptAppInfoInit (ApptDB);
    if (error) return error;
}

// Read the preferences / saved-state information.  There is
// only one version of the DateBook preferences so don't worry
// about multiple versions.
prefsSize = sizeof (DatebookPreferenceType);
```

## Application Control Flow

### *How Events Control an Application*

---

```
if (PrefGetAppPreferences (sysFileCDatebook, datebookPrefID,
                          &prefs, &prefsSize,
                          true) != noPreferenceFound)
{
    DayStartHour = prefs.dayStartHour;
    DayEndHour = prefs.dayEndHour;
    AlarmPreset = prefs.alarmPreset;
    NoteFont = prefs.noteFont;
    SaveBackup = prefs.saveBackup;
    ShowTimeBars = prefs.showTimeBars;
    CompressDayView = prefs.compressDayView;
    ShowTimedAppts = prefs.showTimedAppts;
    ShowUntimedAppts = prefs.showUntimedAppts;
    ShowDailyRepeatingAppts = prefs.showDailyRepeatingAppts;
}

TopVisibleAppt = 0;
CurrentRecord = noRecordSelected;

// Load the far call jump table.
FarCalls.apptGetAppointments = ApptGetAppointments;
FarCalls.apptGetRecord = ApptGetRecord;
FarCalls.apptFindFirst = ApptFindFirst;
FarCalls.apptNextRepeat = ApptNextRepeat;
FarCalls.apptNewRecord = ApptNewRecord;
FarCalls.moveEvent = MoveEvent;

return (error);
}
```

---

## The Event Loop

When startup is complete, the application enters an event loop. While in the loop, the application continuously checks for events on the event queue. If there are events on the queue, the application has to process them as determined in the event loop. As a rule, the events are passed on to the system, which knows how to handle them. For example, the system knows how to respond to pen taps on forms or menus.

The application typically remains in the event loop until the system tells it to shut itself down by sending an [appStopEvent](#) (not a launch code) through the event queue. The application must detect this event and terminate.

### Listing 2.2 Top-Level Event Loop Example

---

```
static void EventLoop (void)
{
    Word error;
    EventType event;
    do
    {
        EvtGetEvent (&event, evtWaitForever);

        PreprocessEvent (&event);

        if (! SysHandleEvent (&event))

            if (! MenuHandleEvent (NULL, &event, &error))

                if (! ApplicationHandleEvent (&event))
                    FrmDispatchEvent (&event);

        #if EMULATION_LEVEL != EMULATION_NONE
            ECApptDBValidate (ApptDB);
        #endif
    }
    while (event.eType != appStopEvent);
}
```

---

## Application Control Flow

### *How Events Control an Application*

---

In the event loop, the application iterates through these steps (see [Figure 2.1](#) and [Listing 2.2](#))

1. Fetch an event from the event queue.
2. Call `PreprocessEvent` to allow the datebook event handler to see the command keys before any other event handler gets them. Some of the datebook views display UI that disappears automatically; This UI needs to be dismissed before the system event handler or the menu event handler displays any UI objects.

Note that not all applications need a `PreprocessEvent` functions. It may be appropriate to call `SysHandleEvent` right away.

3. Call `SysHandleEvent` to give the system an opportunity to handle the event.

The system handles events like power on/ power off, Graffiti input, tapping silk-screened icons, or pressing buttons. During the call to `SysHandleEvent`, the user may also be informed about low-battery warnings or may find and search another application.

Note that in the process of handling an event, `SysHandleEvent` may generate new events and put them on the queue. For example, the system handles Graffiti input by translating the pen events to key events. Those, in turn, are put on the event queue and are eventually handled by the application.

`SysHandleEvent` returns `TRUE` if the event was completely handled, that is, no further processing of the event is required. The application can then pick up the next event from the queue.

4. If `SysHandleEvent` did not completely handle the event, the application calls [MenuHandleEvent](#). `MenuHandleEvent` handles two types of events:
  - If the user has tapped in the area that invokes a menu, `MenuHandleEvent` brings up the menu.
  - If the user has tapped inside a menu to invoke a menu command, `MenuHandleEvent` removes the menu from the screen and puts the events that result from the command onto the event queue.

`MenuHandleEvent` returns `TRUE` if the event was completely handled.

5. If `MenuHandleEvent` did not completely handle the event, the application calls `ApplicationHandleEvent`, a function your application has to provide itself. `ApplicationHandleEvent` handles only the [frmLoadEvent](#) for that event; it loads and activates application form resources and sets the event handler for the active form.
6. If `ApplicationHandleEvent` did not completely handle the event, the application calls [FrmDispatchEvent](#). `FrmDispatchEvent` first sends the event to the application's event handler for the active form. This is the event handler routine that was established in `ApplicationHandleEvent`. Thus the application's code is given the first opportunity to process events that pertain to the current form. The application's event handler may completely handle the event and return `TRUE` to calls from `FrmDispatchEvent`. In that case, calls `FrmDispatchEvent` returns to the application's event loop. Otherwise, calls `FrmDispatchEvent` calls `FrmDispatchEvent` to provide the system's default processing for the event.

For example, in the process of handling an event, an application frequently has to first close the current form and then open another one, as follows:

- The application calls [FrmGotoForm](#) to bring up another form. `FrmGotoForm` queues a [frmCloseEvent](#) for the currently active form, then queues [frmLoadEvent](#) and [frmOpenEvent](#) for the new form.
- When the application gets the `frmCloseEvent`, it closes and erases the currently active form.
- When the application gets the `frmLoadEvent`, it loads and then activates the new form. Normally, the form remains active until it's closed. (Note that this wouldn't work if you preload all forms, but preloading is really discouraged. Applications don't need to be concerned with the overhead of loading forms; loading is so fast that applications can do it when they need it.) The application's event handler for the new form is also established.

## Application Control Flow

### *How Events Control an Application*

---

- When the application gets the `frmOpenEvent`, it performs any required initialization of the form, then draws the form on the display.

After `FrmGotoForm` has been called, any further events that come through the main event loop and to `FrmDispatchEvent` are dispatched to the event handler for the form that's currently active. For each dialog box or form, the event handler knows how it should respond to events, for example, it may open, close, highlight, or perform other actions in response to the event. [FrmHandleEvent](#) invokes this default UI functionality.

After the system has done all it can to handle the event for the specified form, the application finally calls the active form's own event handling function. For example, in the datebook application, it may call `DayViewHandleEvent` or `WeekViewHandleEvent`.

Notice how the event flow allows your application to rely on system functionality as much as it wants. If your application wants to know whether a button is pressed, it has only to wait for [ctlSelectEvent](#). All the details of the event queue are handled by the system.

Some events are actually requests for the application to do something, for example, [frmOpenEvent](#). Typically, all the application does is draw its own interface, using the functions provided by the system, and then waits for events it can handle to arrive from the queue.

Only the active form should process events.

## The Stop Routine

In the stop routine, an application should first flush all active records, then close the application's database, and finally save those aspects of the current state needed for startup. [Listing 2.3](#) is an example of a `StopApplication` routine from `Datebook.c`.

### Listing 2.3 Example of StopApplication Routine

---

```
static void StopApplication (void)
{
    DatebookPreferenceType prefs;

    // Write the preferences / saved-state information.
    prefs.noteFont = NoteFont;
    prefs.dayStartHour = DayStartHour;
    prefs.dayEndHour = DayEndHour;
    prefs.alarmPreset = AlarmPreset;
    prefs.saveBackup = SaveBackup;
    prefs.showTimeBars = ShowTimeBars;
    prefs.compressDayView = CompressDayView;
    prefs.showTimedAppts = ShowTimedAppts;
    prefs.showUntimedAppts = ShowUntimedAppts;
    prefs.showDailyRepeatingAppts = ShowDailyRepeatingAppts;

    // Write the state information.
    PrefSetAppPreferences (sysFileCDatebook, datebookPrefID,
        datebookVersionNum, &prefs, sizeof (DatebookPreferenceType),
        true);

    // Send a frmSave event to all the open forms.
    FrmSaveAllForms ();

    // Close all the open forms.
    FrmCloseAllForms ();

    // Close the application's data file.
    DmCloseDatabase (ApptDB);
}
```

---

## How Launch Codes Control an Application

Launch codes allow direct communication between the system and an application and between two applications. This direct communication takes precedence over any events on an application's queue.

- **The system** uses launch codes to ask an application to do something, interrupting other activities if necessary. Examples include launch codes for launching an application, initializing databases, or resetting after the user performs a hard reset.

Global find is a frequently used launch code that illustrates the usefulness of launch codes. It allows users to search all databases for a certain record, such as a name. In this case, it would be very wasteful to do a full launch—including the user interface—of each application only to access the application's databases in search of that item. Using a launch code avoids this overhead.

- **An application** can use a launch code to request that another application perform an action or modify its data. For example, a data collection application could instruct an email application to queue up a particular message to be sent.

Launch codes can be sent from the system's top level or from another application's thread. In most cases, an application's global variables are not available. Launch codes are delivered to an application at its highest level (through the `PilotMain` function). Each launch code may be accompanied by a [Parameter Block](#) which may in turn contain one or more [Launch Flags](#). The parameter block is specific to the launch code, while the launch flags can be sent with any launch code.

---

**Note:** Static local variables are stored with the global variables on the system's dynamic heap. They are not accessible while executing launch codes other than normal launch.

---

## Parameter Block

Many launch codes are accompanied by a parameter block. A parameter block is a pointer to a structure that contains several parameters. These parameters contain information necessary to handle the associated launch code.

## Launch Flags

Launch flags provide some additional information on what exactly an application should do when it receives a launch code.

- If an application sends a launch code to another application, it should always set the launch flags to zero.
- The system sometimes uses flags with a launch code to indicate how the application should behave. For example, a flag could be used to specify whether the UI should be displayed or not.

Note that even if an application has decided to handle a certain launch code, it can still decide not to handle the associated launch flags.

See [More About Launch Flags](#) for additional information.

## Launch Code Example

An application needs to check for launch codes in its main function. [Listing 2.4](#) shows parts of `PilotMain` from the datebook application as an example. To see the complete example, go to Palm OS SDK: `Examples:Datebook:Datebook.c`.

---

### Listing 2.4 Code Fragment Checking for Launch Codes

---

```
static DWord DBPilotMain (Word cmd, Ptr cmdPBP, Word launchFlags)
{
    Word error;
    Boolean launched;

    // This app makes use of PalmOS 2.0 features. It will crash if
    // run on an earlier version of PalmOS. Detect and warn if this
    // happens, then exit.
    error = RomVersionCompatible (version20, launchFlags);
    if (error)
        return error;

    // Launch code sent by the launcher or the datebook button.
    if (cmd == sysAppLaunchCmdNormalLaunch)
    {
        error = StartApplication ();
        if (error) return (error);

        FrmGotoForm (DayView);
        EventLoop ();
        StopApplication ();
    }

    // Launch code sent by text search.
    else if (cmd == sysAppLaunchCmdFind)
    {
        Search ((FindParamsPtr)cmdPBP);
    }
}
```

```
// This launch code might be sent to the app when it's already
// running if the user hits the "Go To" button in the Find
// Results dialog box.
else if (cmd == sysAppLaunchCmdGoTo)
{
    launched = launchFlags & sysAppLaunchFlagNewGlobals;
    if (launched)
    {
        error = StartApplication ();
        if (error) return (error);

        GoToItem ((GoToParamsPtr) cmdPBP, launched);

        EventLoop ();
        StopApplication ();
    }
    else
        GoToItem ((GoToParamsPtr) cmdPBP, launched);
}

// Launch code sent by sync application to notify the datebook
// application that its database was been synced.
// ...
// Launch code sent by Alarm Manager to notify the datebook
// application that an alarm has triggered.
// ...
// Launch code sent by Alarm Manager to notify the datebook
// application that is should display its alarm dialog.
// ...
// Launch code sent when the system time is changed.
// ...
// Launch code sent after the system is reset. We use this time
// to create our default database if this is a hard reset
// ...
// Launch code sent by the DesktopLink server when it create
// a new database. We will initializes the new database.
return (0);
}
```

---

## Summary of All Launch Codes

The following table lists all Palm OS standard launch codes in alphabetical order. More detailed information is provided immediately after the table (you can also click on the links to access it).

**Table 2.1 Palm OS Launch Codes**

<b>Code</b>	<b>Request</b>
<a href="#">sysAppLaunchCmdAlarmTriggered</a>	Schedule next alarm or perform quick actions such as sounding alarm tones.
<a href="#">sysAppLaunchCmdCountryChange</a>	Respond to country change.
<a href="#">sysAppLaunchCmdDisplayAlarm</a>	Display specified alarm dialog or perform time-consuming alarm-related actions.
<a href="#">sysAppLaunchCmdExgAskUser</a>	Let application override display of dialog asking user if they want to receive incoming data via the exchange manager.
<a href="#">sysAppLaunchCmdExgReceiveData</a>	Notify application that it should receive incoming data via the exchange manager.
<a href="#">sysAppLaunchCmdFind</a>	Find a text string.
<a href="#">sysAppLaunchCmdGoto</a>	Go to a particular record, display it, and optionally select the specified text.
<a href="#">sysAppLaunchCmdInitDatabase</a>	Initialize database.
<a href="#">sysAppLaunchCmdLookup</a>	Look up data. In contrast to <code>sysAppLaunchCmdFind</code> , a level of indirection is implied. For example, look up a phone number associated with a name.
<code>sysAppLaunchCmdNormalLaunch</code>	Launch normally.

**Table 2.1 Palm OS Launch Codes**

<b>Code</b>	<b>Request</b>
<a href="#">sysAppLaunchCmdPanelCalledFromApp</a>	Tell preferences panel that it was invoked from an application, not the Preferences application.
<a href="#">sysAppLaunchCmdReturnFromPanel</a>	Tell an application that it's restarting after preferences panel had been called.
<a href="#">sysAppLaunchCmdSaveData</a>	Save data. Often sent before find operations.
<a href="#">sysAppLaunchCmdSyncNotify</a>	Notify applications that a HotSync has been completed.
<a href="#">sysAppLaunchCmdSystemLock</a>	Sent to the Security application to request that the system be locked down.
<a href="#">sysAppLaunchCmdSystemReset</a>	Respond to system reset. No UI is allowed during this launch code.
<a href="#">sysAppLaunchCmdTimeChange</a>	Respond to system time change.

## More About 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.

The section discusses all launch codes in alphabetical order. For a listing, see [Table 2.1](#).

## **sysAppLaunchCmdAlarmTriggered**

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

### **Impact on Application**

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. An opportunity to perform more time-consuming actions will come when [sysAppLaunchCmdDisplayAlarm](#) is sent.

## **sysAppLaunchCmdCountryChange**

Respond to country change.

### **Impact on Application**

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

## **sysAppLaunchCmdDisplayAlarm**

Perform full, possibly blocking, handling of alarm.

### **Impact on Application**

This is the application's opportunity to handle an alarm in a lengthy or blocking fashion. Notification 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.

## **sysAppLaunchCmdExgAskUser**

Exchange manager sends this launch code to the application when data has arrived for that application. This launch code lets the

application tell the exchange manager whether or not to display a dialog asking the user if they want to accept the data. If the application chooses not to handle this launch command, the default course of action is that the exchange manager displays a dialog asking the user if they want to accept the incoming data. In most cases, applications won't need to handle this launch code, since the default action is the preferred alternative.

The application can respond to this launch code by setting the `result` field in the parameter block to the appropriate value. If it wants to allow the exchange manager to display a dialog, it should leave the `result` field set to `exgAskDialog` (the default value). To disable display of the dialog and to automatically accept the incoming data (as if the user had pressed OK in the dialog), set the `result` field to `exgAskOk`. To disable display of the dialog and to automatically reject the incoming data (as if the user had pressed Cancel in the dialog), set the `result` field to `exgAskCancel`. In the later case, the data is discarded and no further action is taken by the exchange manager.

If the application sets the `result` field to `exgAskOk`, or the dialog is displayed and the user presses the OK button, then the exchange manager sends the application the next launch code, [sysAppLaunchCmdExgReceiveData](#), so that it can actually receive the data.

### **sysAppLaunchCmdExgAskUser Parameter Block**

**Prototype**    `typedef struct {`  
                  `ExgSocketPtr     socketP;`  
                  `ExgAskResultType result;`  
                  `} ExgAskParamType;`

<b>Fields</b>	<code>socketP</code>	Socket pointer
	<code>result</code>	Show dialog, auto-confirm, or auto-cancel

## **sysAppLaunchCmdExgReceiveData**

Following the launch code [sysAppLaunchCmdExgAskUser](#), the exchange manager sends this launch code to the application to notify it 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.

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 will be 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 of the `ExgSocketPtr` data type. It is a pointer to the `ExgSocketType` structure corresponding to the exchange manager connection via which the data is arriving. You will need to pass this pointer to the `ExgAccept` function to begin receiving the data. For more details, refer to the [Exchange Manager](#) chapter in Part III.

## **sysAppLaunchCmdFind**

This launch command is used to implement the global find. It is sent by the system whenever the user enters a text string in a Find dialog. At that time, the system queries each application whether it handles this launch code and returns any records matching the find request.

The system, sends this launch code with the `FindParamsType` parameter block to each application. The system displays the results of the query in the Find dialog.

### Impact on Application

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.

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. See Phase 14 of the tutorial for an example.

### sysAppLaunchCmdFind Parameter Block

```
Prototype typedef struct {  
  
    // These fields are used by the applications.  
    Word          dbAccessMode;  
    Word          recordNum;  
    Boolean       more;  
    Char          strAsTyped [maxFindStrLen+1];  
    Char          strToFind [maxFindStrLen+1];  
    // These fields are private to the Find routine  
    //and should NOT be accessed by applications.  
    Word          numMatches;  
    Word          lineNumber;  
    Boolean       continuation;  
    Boolean       searchedCaller;  
    LocalID      callerAppDbID;  
    Word          callerAppCardNo;  
    LocalID      appDbID;  
    Word          appCardNo;  
    Boolean       newSearch;  
    DmSearchStateType searchState;  
    FindMatchType match [maxFinds];  
} FindParamsType;
```

## Application Control Flow

### More About Launch Codes

---

<b>Fields</b>	<code>dbAccessMode</code>	Read mode. May be “show secret.”
	<code>recordNum</code>	Index of last record that contained a match.
	<code>more</code>	TRUE if more matches to display.
	<code>strAsTyped [maxFindStrLen+1]</code>	Search string as entered.
	<code>strToFind [maxFindStrLen+1]</code>	Search string in lower case.
	<code>numMatches</code>	System use only.
	<code>lineNumber</code>	System use only.
	<code>continuation</code>	System use only.
	<code>searchedCaller</code>	System use only.
	<code>callerAppDbID</code>	System use only.
	<code>callerAppCardNo</code>	System use only.
	<code>appDbID</code>	System use only.
	<code>appCardNo</code>	System use only.
	<code>newSearch</code>	System use only.
	<code>searchState</code>	System use only.
	<code>match [maxFinds]</code>	System use only.

## sysAppLaunchCmdGoto

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

### Impact on Application

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

### sysAppLaunchCmdGoto Parameter Block

**Prototype**

```
typedef struct {
    Word    searchStrLen;
    Word    dbCardNo;
    LocalID dbID;
    Word    recordNum;
    Word    matchPos;
    Word    matchFieldNum;
    DWord   matchCustom;
} GoToParamsType;
```

<b>Fields</b>	searchStrLen	Length of search string.
	dbCardNo	Card number of the database.
	dbID	Local ID of the database.
	recordNum	Index of record containing a match.
	matchPos	Position of the match.
	matchFieldNum	Field number string was found in.
	matchCustom	Application-specific information.

### 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

## Application Control Flow

### More About Launch Codes

---

handling the `sysAppLaunchCmdInitDatabase` command properly. Be especially careful not to access global variables.

#### Impact on Application

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;
    ULong      creator;
    ULong      type;
    UInt       version;
} SysAppLaunchCmdInitDatabaseType;
```

<b>Fields</b>	<code>dbP</code>	Database reference.
	<code>creator</code>	Database creator.
	<code>type</code>	Database type.
	<code>version</code>	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.

#### Impact on Application

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.

### Parameter Block

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

## **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)

### Impact on Application

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.

## 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.

## sysAppLaunchCmdSaveData

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

### Impact on Application

Any application that supports the Find command and that can have buffered data should support this launch code. Generally, an application only has to respond if it's the currently running application. In that case, all buffered data should be saved when the launch code is received.

### sysAppLaunchCmdSaveData Parameter Block

**Prototype**

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

**Fields**

uiComing	TRUE if system dialog is displayed before launch code arrives.
----------	--

## sysAppLaunchCmdSyncNotify

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

## sysAppLaunchCmdSystemLock

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

### Impact on Application

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

## sysAppLaunchCmdSystemReset

Launch code to respond to system soft or hard reset.

### Impact on Application

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 the Palm OS Cookbook.

### sysAppLaunchCmdSystemReset Parameter Block

**Prototype**

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

**Fields**

<code>hardReset</code>	TRUE if system was hardReset. FALSE if system was softReset.
<code>createDefaultDB</code>	If TRUE, application has to create default database.

## **sysAppLaunchCmdTimeChange**

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

### **Impact on Application**

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.

## **More About Launch Flags**

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

An application may decide not to handle the flags even if it handles the launch code itself. For applications that decide to include this launch code, the following table provides additional information:

<b>Flag</b>	<b>Functionality</b>
sysAppLaunchFlagNewThread	Creates a new thread for the application. Implies sysAppLaunchFlagNewStack.
sysAppLaunchFlagNewStack	Creates a separate stack for the application.
sysAppLaunchFlagNewGlobals	Creates a new globals world for the application. Implies new owner ID for memory chunks.

<b>Flag</b>	<b>Functionality</b>
sysAppLaunchFlagUIApp	Notifies launch routine that this is a UI application being launched.
sysAppLaunchFlagSubCal	Notifies launch routine that 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.

Generally, the system sends launch flags along with all launch codes. Applications should just pass 0 (zero) when sending a launch code to another application.

## Responding to Launch Codes

Launch codes may be sent to any application without negative effects. However, a launch code only has an effect if the application that receives it has been programmed to handle it. An application may decide not to handle the flags even if it handles the launch code itself.

When developing your application, be sure to handle as many of the standard launch codes as possible.

When an application receives a launch code, it must first check whether it can handle this particular code. For example, only applications that have text data should respond to a launch code requesting a string search. If an application can't handle a launch code, it exits without failure. Otherwise, it performs the action immediately and returns.

## Determining Status When Receiving Launch Code

If an application receives a launch code other than `sysCmdAppNormalLaunch`, it can find out whether it's the current application by checking the launch flags, which are sent to the currently running application.

If the application is the currently running application, the `sysAppLaunchFlagSubCall` flag is set. This flag is set by the system and isn't (and shouldn't be) set by the sender of a launch code.

Note that if the launch code is sent to the currently running application, the launch code handler may access the application's global variables. Only the system can access these global variables.

## Predefined Launch Codes

A number of launch codes are predefined by the system for handling certain system tasks, for example,

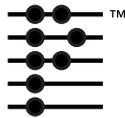
- Notifying the application when certain system preferences like date and time have changed
- Performing global find and goto operations
- Notifying the application that its data files have been updated by a sync operation

The launch code parameter is a 16-bit word value. All launch codes with values 0–32767 are reserved for use by the system and for future enhancements. Launch codes 32768–65535 are available for private use by applications.

## Creating Your Own Launch Codes

In addition to the predefined launch codes defined in [Table 2.1](#), developers can create their own launch codes to implement specific functionality. Both the sending and the receiving application must know about and handle any developer-defined launch codes.

An example is `PhoneLookup.c`.



# Palm OS User Interface Resources

---

Palm OS User Interface resources are the elements of an application's GUI (graphical user interface).

This chapter helps you work with resources by providing information about these topics:

- [Using Constructor to Work With Resources](#) gives an overview of the Constructor tool and briefly explains how to use it.
- [Project Resources](#) provides information about project resources. These resources are created by instantiating a template. Examples are menu and menu bar or string.
- [Catalog Resources](#) provides information about resources that you can instantiate by dragging them from the catalog onto a form. Examples are buttons and check boxes.

---

**Note:** For more information see the following manuals:

The [Palm OS Tutorial](#) provides more detailed instruction on how to create a GUI using the Constructor tool.

The [Constructor for Palm OS](#) manual in the CodeWarrior Documentation folder provides detailed reference-style documentation as well as information on how to use each individual resource.

---

The relationship between the resources and the structures provided by Palm OS is discussed in [Chapter 4, “Palm OS User Interface Objects.”](#)

## Using Constructor to Work With Resources

In Palm OS 2.0 and later, developers can choose how they want to create their resources:

- Using Constructor. This chapter describes how to use Constructor to create and manipulate resources.
- Macintosh users can also use ResEdit (as under Palm OS 1.0)

### Creating Resources

Constructor has a graphical interface that allows you to quickly create and view a GUI for your application. Here's an overview of how it works. For detailed information, see the "Palm OS Tutorial."

**1. Open Constructor.**

Constructor opens a catalog window that contains all catalog resources such as buttons, check boxes, or tables. If the catalog window isn't visible, you can type Cmd-Y to display it.

**2. From the File Menu, choose New Project File or Open Project File.**

Constructor opens the Constructor project window that lets you instantiate project resources such as forms, menus, and strings and specify project settings.

**3. In the Constructor project window, select the Forms template and type Cmd-K.**

Constructor instantiates the form; an icon representing it appears below the template.

**4. Double-click on the form.**

Constructor opens a Form Editor, with information about the resource (left panel) and a graphical representation (right panel).

**5. Drag the additional desired UI elements from the catalog to the Forms window**

After you've dragged a resource icon onto the form, the resource information becomes visible in the left panel (Layout Properties).

**6. Instantiate special resource types, such as strings or bitmaps, in the project window and associate them with the related resource.**

## Changing Resources

You can make changes to any resource as follows:

- To change a project resource, double-click on that resource in the project window and change the fields in the associated editor that appears.
- To change a catalog resource, you have several choices:
  - Move any UI element in the Layout Appearance panel of the Forms window.
  - Change the values in the left (Layout Properties) panel of the Forms window.
  - Double-click the UI element and change the values in the Inspector that appears.

## Project Resources

Catalog resources are available in the Catalog window and can be dragged directly on a form. All other resources, including the form itself, are instantiated from the projects window.

The following table lists all Palm OS project resources in alphabetical order by resource name. The Macintosh ResEdit resource name is include for reference only; it's not needed by developers who use Constructor exclusively, and not relevant for Windows developers.

<b>Name</b>	<b>Resource</b>	<b>UI Name</b>
Talt	<a href="#">Alerts</a>	Alert
tFRM	<a href="#">Form Resource</a>	Form
	Menu Resource	Menu
	Menu bar Resource	Menu bar
tSTR	<a href="#">String Resource</a>	String
	Icons	
	Bitmaps	

---

## Alerts

### Example



**Overview** The alert resource defines a modal dialog that displays a message, an icon, and one or more buttons.

A small icon indicates the category of the dialog box; for example, an exclamation mark for an error message. The icon appears on the left side of the dialog. The text is justified left but placed to the right of the dialog icon.

Type	Icon	Definition	Button	Example
Information	i	Lowest-level warning. Action shouldn't or can't be completed but doesn't generate an error or risk data loss.	OK	An alarm setting must be between 1 and 99.
Confirmation	?	Confirm an action or suggest options.	OK, Cancel	Change settings before switching applications? (For example, when pressing an application key with an open dialog box.)
Warning	!	Ask if user wishes to continue a potentially dangerous action.	OK, Cancel	Are you sure you want to delete this entry?
Error	(stop sign)	Attempted action generated error and/or cannot be completed.	OK	Disk full.

## Palm OS User Interface Resources

### Alerts

---

The Alert resource has the following attributes.

<b>Attributes</b>	<b>Alert Type</b>	Determines the sound played and the icon displayed when the alert is drawn. There are four possible icons: <ul style="list-style-type: none"><li>• InformationAlert (Alert Number 0)</li><li>• ConfirmationAlert (Alert Number 1)</li><li>• WarningAlert (Alert Number 2)</li><li>• ErrorAlert (Alert Number 3)</li></ul>
	<b>Help ID</b>	The ID of a String resource that's the help text for the alert dialog box. If you provide a value, the system displays an "i" in the top right corner of the alert box.
	<b>Default Button ID</b>	The number of a button that the system assumes is selected if the user switches to another application, forcing the form to go away without making a selection.
	<b>Title</b>	Title of the alert form.
	<b>Message</b>	Message displayed by the alert dialog. May contain ^1, ^2, ^3 as substitution variables for use in conjunction with <a href="#">FrmCustomAlert</a> .
	<b>Button Text</b>	Text of the button (e.g. OK or Cancel), determined by an entry in the resource of each button. To add a button, select Item Text 0, and type Cmd-K.

## Form Resource

**Overview** A form is a container for one or more of the [Catalog Resources](#).

Applications usually contain several different forms that the user triggers by tapping buttons or other control UI objects. Most UI objects are displayed only if they are contained within a form.

**Example** The example below shows a modal form. A form can also be as large as the screen.



<b>Attributes</b>	Left Origin	Window-relative position of left side of form. Valid values: 0 – 159
	Top Origin	Window-relative position of top of form. Valid values: 0 – 159
	Width	Width of the form in pixels. Valid values: 0 – 160
	Height	Height of the form in pixels. Valid values: 1– 160
	Usable	Not currently supported for forms.
	Modal	If checked, form is modal. Modal forms ignore pen events outside their boundaries. Used for dialogs.
	Save Behind	If checked, the region obscured by the form is saved when it's drawn and restored when it's erased. Used for dialogs.

## Palm OS User Interface Resources

### Form Resource

---

Form ID	Form ID assigned by Constructor.
Help ID	ID number of a string that's displayed when the user taps the "i" icon. The system adds the icon to the form when you provide a value for this property. Currently, only modal dialogs have help resources.
Menu Bar ID	Contains the ID of a menu bar resource to be associated with this forms.
Default Button ID	Number of a button that the system assumes is selected if the user switches to another application, forcing the form to go away without making a selection.
Form Title	Title of that form. Use titles for dialogs, menu bars for views. By convention, the name of the application and the name of the screen, if possible, for example Address List or Address Edit. The title must be one line; it uses about 13 pixels of the top of the form.
Palm OS Version	Version of the device for which this form is created.

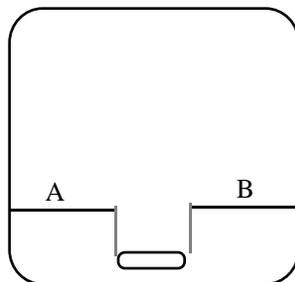
**Comments** The total display on the Palm device is 160 pixels by 160 pixels. If you want your whole form to be seen, make sure it fits within this display area. For pop-up dialogs, you can make the form smaller. Align a popup dialog with the bottom of the screen.

A form is the GUI area for each view of your application. For example the Address Book offers an Address List view, Address Edit view, and so on. Each application has to have one form, and most applications have more than one. To actually create the view, you have to add other UI elements to the form; either by dragging them onto the form from the catalog or by providing their ID as the value of some of the form's fields.

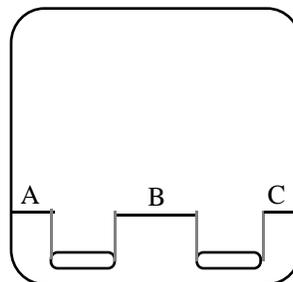
Here are some general design guidelines:

- Each form should have a title that displays the name or view of the application, or both.
- Scroll bars in fields and tables appear and disappear dynamically if you've selected that option for that UI element. Place them to the right of command buttons.
- Modal dialogs always occupy the full width of the screen and are justified to the bottom of the screen. They hide the command buttons of the base application but don't obscure the title bar of the base application if possible. There should be a minimum of three pixels between the top of the modal dialog title bar and the bottom of the application title bar. If the dialog is too large to accommodate this, the entire application title bar should be obscured.
- Screen command buttons should always be at the bottom of the screen.
- Dialog command buttons appear four pixels above the bottom of the dialog box frame. Two-pixel default ring is three pixels above the bottom, and the baseline of the text within the buttons should be aligned.
- Command buttons should be centered so that the spaces between the buttons are twice the width of the spaces between the edges and the border (see diagram below).

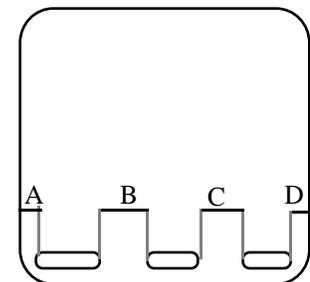
If possible, all buttons should be the same width. At a minimum, they should be spaced equidistant, as illustrated below.



One button  
Distance  $A = B$



Two buttons  
Distance  $B = 2 \times A = 2 \times C$



Three buttons  
Distance  $B = C = 2 \times A = 2 \times D$

**Event Flow** When a form is opened, a [frmOpenEvent](#) is triggered and the form's ID is stored. A [winEnterEvent](#) is triggered whenever a form is opened, and a [winExitEvent](#) is triggered whenever a form is closed.

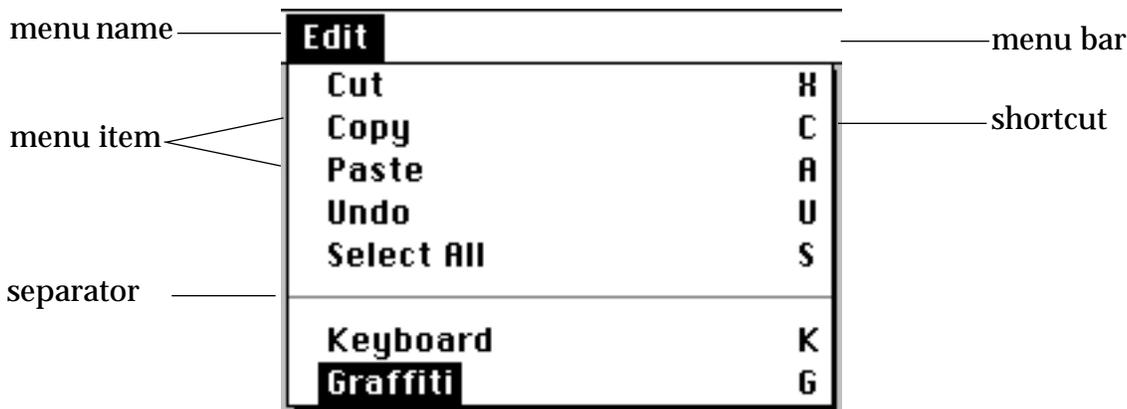
## String Resource

<b>Name</b>	Strings
<b>Overview</b>	Stores data strings used by the program. String resources may be entered as text strings or as a series of hexadecimal characters.
<b>Attributes</b>	String      The text string to be stored, in decimal ASCII.
<b>Comments</b>	The string resource uses either the string or data. If both are entered, they are concatenated.

## Menus and Menu Bars

This section first provides a [Menu Overview](#), then steps you through [Creating a Menu](#). This is followed by a discussion of the two resources: [Menu Bar and Menu Resources](#). Finally, you learn about [Menu User Interaction](#) and [Event Flow for Menu Resource](#).

### Menu Overview



A menu assembly consists of a menu bar, menu names indicating the available menus, and the menus themselves with their commands:

- **Menu bar.** The menu bar at the top of the screen contains the names of the available menus. Each application has different sets of menu names; within an application, different views may have different menus.
- **Menu name.** Each menu is displayed below the menu name. The following menu names are commonly found:
  - Record—Place Record to the left of Edit (if applicable).
  - Edit—Screens that allow editing need an Edit menu. Note, however, that most editing is edit-in-place.
  - Options—Typically, the last menu. The About command, which provides version and creator information, should always be an Options command under Palm OS.
- **Menu.** The menus themselves consist of menu items and optional shortcuts. Under Palm OS, menu items should **not** duplicate functionality available via command buttons. Menus justify left with the active heading of the menu name when invoked. If the menu doesn't fit, it's justified to the right border of the screen.

---

**Note:** For each menu, provide shortcuts for all commands or for none at all. Don't assign the same shortcut twice within one application.

---

## Creating a Menu

In Palm OS 2.0 and later, you can interactively create the menu bar and all menus, then associate the menu bar with the form.

To create a menu assembly using Constructor, follow these steps:

1. **In the project window, select Menu Bars, then type Cmd-K.**  
Constructor creates a menu bar instance.
2. **Name the Menu Bar instance, then double-click on it.**  
Constructor opens the Menu Editor.

## Palm OS User Interface Resources

### Menus and Menu Bars

---

3. **Back in the project window, create one or more Menus, name them, and drag them onto the Menu Editor.**
  4. **For each menu, replace the “untitled” default text with the menu name, such as Edit or File.**
  5. **Type Cmd-K to add menu items, Cmd- (minus) to add separators to the menu.**
  6. **To assign a shortcut key, you can do one of the following:**
    - Tab from the menu item to the shortcut region in the menu editor, then enter the shortcut letter.
    - Type Cmd-I and enter the shortcut in the property inspector that appears.
- The system will later add the shortcut symbol before the character.
7. **When you’re finished with the assembly, close the window.**
  8. **Finally, enter the ID of the menu bar you created into the Menu Bar ID property of a form.**

---

**Note:** The Palm OS Tutorial provides more detailed step-by-step instructions for creating a menu.

---

## Menu Bar and Menu Resources

The only information provided for the menu and menu bar resource is the resource name and resource ID.

## Menu User Interaction

- **Default Menu and Menu Item.** A pen-up on the menu icon displays the menu bar. The first time a menu is invoked after an application is launched, no menus are displayed unless there is only one menu available. Afterwards the menu and menu item of the last command executed from the menu are displayed. Graffiti command equivalents are ignored.

For example, if the user selects Edit > Copy, the Edit menu is popped down and the Copy command is highlighted the next time the menu bar is displayed. This expedites execution of commonly used commands or of grouped commands (e.g., Copy/

Paste). The last menu heading is not saved if the user switches to a different view or a different application.

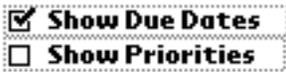
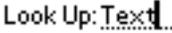
- **View-specific Menus.** Each view within an application can have a unique menu, that is, different menu headings and items.
- **Menu Display.** As a rule, a Palm OS application should try to have the menu visible on screen as rarely as possible:
  - After a menu command is executed, the menu bar is dismissed.
  - The menu bar is active when the menu headings in it are active. When not active, the menu bar is not visible.
  - There are no grayed-out menu headings or grayed-out menu items. A command not accessible in a certain mode doesn't appear at all.
- **Size.** The vertical active area of menu headings is 2 pixels beyond the ascender and 1 pixel below a potential descender of the menu heading text. The horizontal active area covers half the distance to the next menu heading, leaving no gaps between the headings. If the menu headings aren't as wide as the menu bar, part of the menu bar may be inactive.
- **Active Area.** The entire area of the menu, excluding the border, is active. Divider lines and status items on the launcher menu are inactive; that is, they do not highlight when tapped.

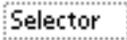
## Event Flow for Menu Resource

User Action	System Response
Pen enters menu window.	<a href="#">winExitEvent</a> to exit previous window. <a href="#">winEnterEvent</a> to enter menu window. <a href="#">penDownEvent</a> is also triggered, although the pen has not actually touched the screen.
User selects a menu item.	WinExitEvent to exit menu window. WinEnterEvent to enable the form the menu spawned. <a href="#">menuEvent</a> (store ID number of the item in EventType). <a href="#">penUpEvent</a> finally occurs.

## Catalog Resources

You can add Constructor catalog resources to the user interface by dragging the corresponding icon onto a Form. The following catalog resources are available:

Name	Resource	Resource
tBTN	<a href="#">Button Resource</a>	
tCBX	<a href="#">Check Box Resource</a>	
tFLD	<a href="#">Field Resource</a>	
tFBM	<a href="#">Form Bitmap Resource</a>	(container for Bitmap resource)
tGDT	<a href="#">Gadget Resource</a>	(application defined)

Name	Resource	Resource
tGSI	<a href="#">Graffiti Shift Indicator Resource</a>	↑
tLBL	<a href="#">Label Resource</a>	(container for a String)
tLST	<a href="#">List Resource</a>	
tPUT	<a href="#">Popup Trigger Resource</a>	▼ Work
tPBN	<a href="#">Push Button Resource</a>	: 1 2 3 4 5
tREP	<a href="#">Repeating Button Resource</a>	
	<a href="#">Scrollbar Resource</a>	(see below)
tSLT	<a href="#">Selector Trigger Resource</a>	
tTBL	<a href="#">Table Resource</a>	

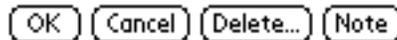
---

## Button Resource

**UI Structure**    ControlType

**Overview**    A button is a clickable UI object, often used to trigger events in an application. A button displays as a text label surrounded by a rectangular frame. The frame has rounded corners. The label may be regular text or a glyph from one of the symbol fonts provided with your development environment, for example, an arrow.

### Examples



<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Button ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of button. Valid values: 0 – 159
	Top Origin	Form-relative position of top of object. Valid values: 0 - 159
	Width	Width of button in pixels. Size the buttons to allow 3–6 pixels of white space at each end of the label. Valid values: 0 – 160
	Height	Height of the button in pixels. Should be 3 pixels larger than the font size, for example, height = 12 for 9-point labels. Valid values: 1 – 160

Usable	A nonusable object is not considered part of the application's interface and doesn't draw. Nonusable objects can programmatically be set to usable. If checked, the object is usable.
Anchor Left	Controls how the object resizes itself when its text label is changed. If checked, the left bound of the object is fixed; if unchecked, the right bound is fixed.
Frame	If checked, a rectangular frame with rounded corners is drawn around the button. Most buttons have frames. Buttons whose labels are single symbol characters, such as scroll buttons, don't have frames.
Non-bold Frame	If checked, a one-pixel-wide rectangular frame with rounded corners is drawn around the button. If unchecked, a bold frame (two pixels wide) is drawn around the button. Non-bold frames are the default.
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text displayed inside the button: one line of text, or a single character from a symbol font to create an increment arrow.

**Comments** The label is centered in the button. If the label text is wider than the button, the whole label is centered and both the right and left sides are clipped.

Place command buttons at the bottom of table views and dialog boxes. Leave three pixels between the dialog bottom and buttons.

Increment arrows are a special case; they are buttons that let users increment the value displayed in a data field.

To create an increment arrow, use an arrow character from the Symbol font as a label. Several arrow styles and sizes are available.

## Palm OS User Interface Resources

### Button Resource

---

#### Event Flow for Button Resource

User Action	System Response
Pen goes down on a button.	<a href="#">penDownEvent</a> ; store x and y coordinates in <code>EventType</code> . <a href="#">ctlEnterEvent</a> ; store button ID number in <code>EventType</code> .
Pen is lifted from button.	<a href="#">ctlSelectEvent</a> ; store button ID number in <code>EventType</code> . <code>ctlSelectEvent</code> can be triggered only if a <code>ctlEnterEvent</code> with the same button ID has just occurred. <a href="#">penUpEvent</a> ; store x and y coordinates in <code>EventType</code> .
Pen is lifted outside button.	Nothing happens.

#### Tip Making a Button with a Bitmap Label

It's not possible to make a bitmap the label of a button; the label always has to be a text string. However, the same effect can be achieved by

- Creating a bitmap the same size of a button
- Placing them at the same location.

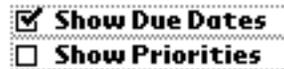
Make sure the bitmap is a Form Bitmap, selected from the catalog. When the user selects the button, the system inverts the bitmap graphic as well.

## Check Box Resource

**UI Structure** ControlType

**Overview** A check box is a small, square UI object with an optional text label to the right.

**Example** The figure below shows a checked and an unchecked check box with a label to the right (the default).



<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Check Box ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
	Top Origin	Form-relative position of top of object. Valid values: 0 – 159
	Width	Width of the picking area around the check box. Valid values: 0 – 160
	Height	Height of the picking area around the check box. Valid values: 1– 160
	Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.

## Palm OS User Interface Resources

### Check Box Resource

---

Selected	Initial selection state of the checkbox. If the box is checked (the default), the checkbox is initially checked.
Group ID	Group ID of a check box that is part of an exclusive group. Ungrouped (nonexclusive) check boxes have 0 as a group ID. Valid values: 0 – 65535
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text displayed to the right of the check box. This text is part of the activation area. To create a (non-active) label to the left of the check box, leave this attribute blank and create a separate Label resource.

**Comments** Make sure that only one check box in a group is initially checked.

All check boxes are the same size. The Height and Width determine the toggle area, which is the screen area the user needs to press to check or uncheck the box.

If a label attribute is defined, it's part of the activation area.

## Event Flow for Check Box Resource

User Action	System Response
Pen goes down on check box.	<p><a href="#">penDownEvent</a>; store x and y coordinates in <code>EventType</code>. <a href="#">ctlEnterEvent</a>; store check box's ID number in <code>EventType</code>.</p> <ul style="list-style-type: none"><li>• If the check box is unchecked, a check appears.</li><li>• If the check box is already checked and is grouped, there is no change in appearance.</li><li>• If the check box is already checked and is ungrouped, the check disappears.</li></ul>
Pen is lifted from check box.	<p><a href="#">ctlSelectEvent</a>; store check box's ID number in <code>EventType</code>, switch check box on (1) or off (0) internally. A <code>ctlSelectEvent</code> can be triggered only if a <code>ctlEnterEvent</code> with the same check box ID number has just occurred. <a href="#">penUpEvent</a>; store x and y coordinates in <code>EventType</code>.</p>
Pen is lifted outside box.	Nothing happens.

## Field Resource

**UI Structure**    `FieldType`

**Overview**    The field UI object is for user data entry in an application. It displays one or more lines of editable text. A field can be underlined, justified left or right, and selectable or unselectable.

Text fields can be located anywhere but in menus and in the command button area.

The following is an underlined, left-justified field containing data:

Look Up: Text.....

<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Field ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
	Top Origin	Form-relative position of top of object. Valid values: 0 – 159
	Width	Width of the object in pixels. Valid values: 0 – 160
	Height	Height of the object in pixels. Valid values: 1– 160
	Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.

Editable	Noneditable fields don't accept user input but can be changed programmatically. If this box is checked, the field is editable.
Underline	If set, each line of text is underlined with a gray line.
Single Line	If checked, the field doesn't scroll horizontally and doesn't accept Return or Tab characters. Only a single line of text is displayed. If the user attempts to enter text beyond this, the system beeps.  Multiline text fields expand. An empty field may display one or more blank lines; for example, records in a To Do list or a text page.
Dynamic Size	If checked, the height of the field is expanded or compressed as characters are added or removed. Set this attribute to false if the Single Line attribute is set.
Left Justified	Text justification. Supported only for fields that have the Single Line attribute checked. Valid values: checked (left-justified)—recommended unchecked (right-justified)
Max characters	Maximum number of characters the field accepts. This is a limit on the number of characters a user can enter, but not on what can be displayed. All fields can display up to 32,767 characters regardless of this setting. Valid values: 0 – 32767
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.

## Palm OS User Interface Resources

### Field Resource

---

Auto-Shift	If checked, 2.0 (and later) auto-shift rules are applied. The system automatically uses an upper-case letter: <ul style="list-style-type: none"><li>• after an empty field</li><li>• after a period or other sentence terminator (e.g. ? or !).</li><li>• after two spaces</li></ul>
Has Scroll-bar	If checked, the system sends more frequent fld-HeightChangedEvents so the application can adjust the height appropriately.

### Event Flow for Field Resource

User Action	System Response
Pen goes down on a field.	<a href="#">penDownEvent</a> ; store x and y coordinates in <code>EventType</code> . <a href="#">fldEnterEvent</a> ; store the field's ID number in <code>EventType</code> .
Pen is lifted.	<a href="#">penUpEvent</a> ; store x and y coordinates in <code>EventType</code> . A field remains selected until another field is selected or the form that contains the field is closed.
User enters characters into selected field.	<a href="#">keyDownEvent</a> ; store ASCII value in <code>EventType</code> .

## Form Bitmap Resource

**Overview** Places predefined bitmaps on a given form. Used for icons in Alert dialogs to indicate a warning, error, information, and so on. You have to associate a Bitmap with the Form Bitmap to actually make a picture appear.

<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Left Origin.	Left bounds of bitmap.
	Top Origin	Top bounds of bitmap.
	Bitmap Resource ID	ID of a PICT resource containing the graphic. You can also assign an ID number, then click on Create and draw the picture in the bitmap editor that appears.
	Usable	Checked if the bitmap should be drawn.

## Gadget Resource

**Name** tGDT

**UI Name** Gadget

**Overview** A gadget object lets developers implement a custom UI gadget. The gadget resource contains basic information about the custom gadget, which is useful to the gadget writer for drawing and processing user input.

<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Gadget ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
	Top Origin	Form-relative position of top of object. Valid values: 0 – 159
	Width	Width of the gadget in pixels. Valid values: 0 – 160
	Height	Height of the gadget in pixels. Valid values: 1– 160
	Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.

## Graffiti Shift Indicator Resource

**Overview** The Graffiti Shift Indicator resource specifies the window-relative or form-relative position of the Graffiti shift state indicator. The different shift states are punctuation, symbol, uppercase shift, and uppercase lock. These indicators will appear at the position of the Graffiti Shift resource.

**Note:** By convention, Graffiti Shift indicators are placed at the bottom-right of every form that has an editable text field.

<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
	Top Origin	Form-relative position of top of object. Valid values: 0 – 159
	Object ID	ID of the object (assigned by Constructor).

## Label Resource

**Overview** The label resource displays noneditable text or labels on a form (dialog box or full-screen). It's used, for example, to have text appear to the left of a checkbox instead of the right.

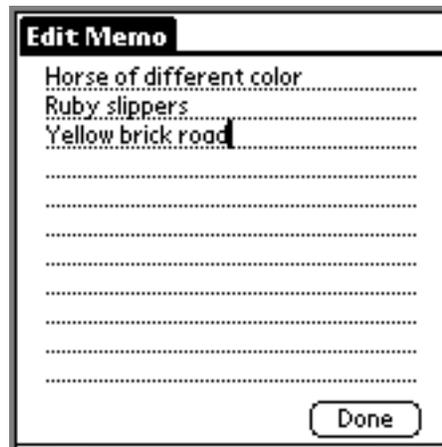
**Comments** Pressing Return in a label wraps the text to the next line.

<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Label ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of object. Valid values: 0 – 159
	Top Origin	Form-relative position of top of object. Valid values: 0 – 159
	Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
	Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
	Text	Text of the label.

## List Resource

**UI Structure**    ListType

### Example



**Overview**    A list provides a box with a list of choices to the user. The list is scrollable if the choices don't all fit in the box.

The list box appears as a vertical list of choices surrounded by a rectangular frame. The current selection of the list is inverted. Arrows for scrolling the list appear in the right margin if necessary.

Lists can appear as popup lists when used with popup triggers. See [Popup Trigger Resource](#).

<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	List ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of object. Valid values: 0 – 159

## Palm OS User Interface Resources

### List Resource

---

Top Origin	Form-relative position of top of object. Valid values: 0 – 159
Width	Width of the list. Valid values: 0 – 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Visible items	Height of the list box, in items (choices). For example, if the list has six items but only four fit, specify four.
Items	Items in the list.

**Comments** Errors may occur if the number of visible items is greater than the actual number of items. An item's text is not clipped against the list box's borders. Set a list to not usable if it's linked to a popup trigger.

Use a list to let users choose between items of data; use a menu to activate a command.

If a list becomes too tall to fit below the trigger, it's justified up. If it becomes too large for the screen, it scrolls.

### Event Flow for List Resource

User Action	System Response
Pen goes down on a list box.	<a href="#">penDownEvent</a> ; store x and y coordinates in <code>EventType</code> . <a href="#">lstEnterEvent</a> ; store list ID and selected item in <code>EventType</code> .
Pen is lifted from the list box.	<a href="#">lstSelectEvent</a> is triggered; store button's ID number and number of selected item in <code>EventType</code> . <a href="#">penUpEvent</a> ; store x and y coordinates in <code>EventType</code> .

## Popup Trigger Resource

**UI structure** `ControlType`

**Overview** The popup trigger shows the selection of a list. The user can press the popup trigger to pop up the list and change the selection.

A popup trigger displays a text label and a triangle to the left of the label that indicates the object is a popup trigger.

When the user selects a popup trigger, a list of items pops up.

▼ Work

<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Popup ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of button. Valid values: 0 – 159
	Top Origin	Form-relative position of top of button. Valid values: 0 – 159
	Width	Width of the button's picking area in pixels. Valid values: 1 – 160
	Height	Height of the button's picking area in pixels. Valid values: 1 – 160
	Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.

## Palm OS User Interface Resources

### Popup Trigger Resource

---

Left anchor	Controls how the object resizes itself when its text label is changed. Valid values: checked (left bound fixed) unchecked (right bound fixed)
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text displayed in the popup trigger (right of arrow).
List ID	ID of the List object that pops up when the user taps the pop-up trigger.

### Event Flow for Popup Trigger Resource

User Action	System Response
Pen goes down on popup trigger.	<a href="#">penDownEvent</a> ; store x and y coordinates in EventType. <a href="#">ctlEnterEvent</a> ; store popup trigger ID number in EventType.
Pen is lifted from popup trigger.	<a href="#">ctlSelectEvent</a> ; store popup trigger ID number in EventType. A <code>ctlSelectEvent</code> can be triggered only if a <code>ctlEnterEvent</code> with the same popup trigger ID number has just occurred. <a href="#">winExitEvent</a> ; pass control to a popup list object.
Popup list pops up.	<a href="#">winEnterEvent</a> <a href="#">penUpEvent</a> ; a <code>penDownEvent</code> to pop up the popup list.

User Action	System Response
Pen goes down on item in popup list.	<code>penDownEvent</code> occurs.
Pen is lifted from popup list.	<a href="#">lstSelectEvent</a> ; store the popup list ID and the selected item number in <code>EventType</code> . <code>winExitEvent</code> causes popup list to disappear; control passes back to the popup trigger. <code>winEnterEvent</code> occurs. <code>popSelectEvent</code> is triggered if an item was selected in the popup list; store popup trigger ID, the popup list ID, and the item number selected in <code>EventType</code> . <code>penUpEvent</code> occurs.

## Push Button Resource

**UI Structure** `ControlType`

**Overview** Push buttons allow users to select an option from a group of items. The choices should have few characters; if the choices are long; check boxes are preferable.

Push buttons display a text label surrounded by a 1-pixel-wide rectangular frame. They appear in a horizontal or vertical row with no pixels between the buttons. The buttons share a common border so there appears to be a one pixel line between two controls. The current selection is highlighted.

Priority:  1  2  3  4  5

Sort by:  Priority  Due Date

The List By dialog of the Address Book and the Details dialog of the ToDo List contain examples of rows of push buttons.

## Palm OS User Interface Resources

### Push Button Resource

---

<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation/update.
	Button ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of button. Valid values: 0 – 159
	Top Origin	Form-relative position of top of button. Valid values: 0 – 159
	Width	Width of the button in pixels. Should be size of label plus two pixels at each end. Valid values: 1 – 160
	Height	Height of the button in pixels. Should be font size plus two pixels. Valid values: 1 – 160
	Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
	Group ID	Group ID of a push button that is part of an exclusive group. Only one push button in an exclusive group may be depressed at a time. Ungrouped (nonexclusive) push buttons have zero as a group ID. This feature must be enforced by the application. Valid values: 0 – 65535
	Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
	Label	Text displayed inside the push button.

**Comment** To create a row of push buttons, create a number of individual push button resources with the same height and align them by specifying the same top position for each button.

## Event Flow for Push Button Resource

User Action	System Response
Pen goes down on push button.	<a href="#">penDownEvent</a> ; store x and y coordinates in <code>EventType</code> . <a href="#">ctlEnterEvent</a> ; store push button ID number in <code>EventType</code> . Push button is highlighted. If push button is grouped and highlighted, no change. If push button is ungrouped and highlighted, it becomes unhighlighted.
Pen is lifted from push button.	<a href="#">ctlSelectEvent</a> ; store button ID number and its current state; on = 1; off = 0. <code>ctlSelectEvent</code> can be triggered only if a <code>ctlEnterEvent</code> with the same push button ID number just occurred. <a href="#">penUpEvent</a> ; store the x and y coordinates.

## Repeating Button Resource

**Overview** The repeating button object is identical to the button object in its appearance. The repeating button is used for buttons that need to be triggered continuously by holding the pen down on them. A good example for a repeating button is the scroll arrow, which moves text as long as it's held down.

<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Button ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of button. Valid values: 0 – 159
	Top Origin	Form-relative position of top of button. Valid values: 0 – 159
	Width	Width of the button in pixels. Valid values: 1 – 160
	Height	Height of the button in pixels. Valid values: 1 – 160
	Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
	Anchor Left	Controls how the object resizes itself when its text label is changed. If checked, the left bound of the object is fixed; if unchecked, the right bound is fixed.
	Frame	If checked, a rectangular frame with rounded corners is drawn around the button.

Non-bold Frame	Determines the width of the rectangular frame drawn around the object. Valid values: checked (1-pixel-wide frame) unchecked (2-pixel-wide frame)
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text displayed inside the button.

**Comments** The attributes match those of the [Button Resource](#) (tBTN); the behavior differs.

You can also use repeating buttons to create increment arrows. See [Button Resource](#) for more information.

## Event Flow for Repeating Button Resource

A repeating button is similar in appearance to a button, but it generates different events. A button generates a [ctlEnterEvent](#) when it is pressed and a `ctlSelect` event when it is released. A repeating button generates a `ctlEnterEvent` when it is pressed and a [ctlRepeatEvent](#) as long as it remains pressed. Here's a more detailed discussion of the events:

User Action	System Response
Pen goes down on a repeating button.	<a href="#">penDownEvent</a> ; store x and y coordinates in <code>EventType</code> . <code>ctlEnterEvent</code> ; store button's ID number in <code>EventType</code> .
Pen remains on repeating button.	For every given amount of time the pen is down on the repeat control object, a <code>ctlRepeatEvent</code> is generated.
Pen is dragged off the repeating button.	No additional <code>ctlRepeatEvent</code> occurs.

**User Action**

**System Response**

Pen is dragged back onto the button.

[ctlRepeatEvent](#) begins to occur again.

Pen is lifted.

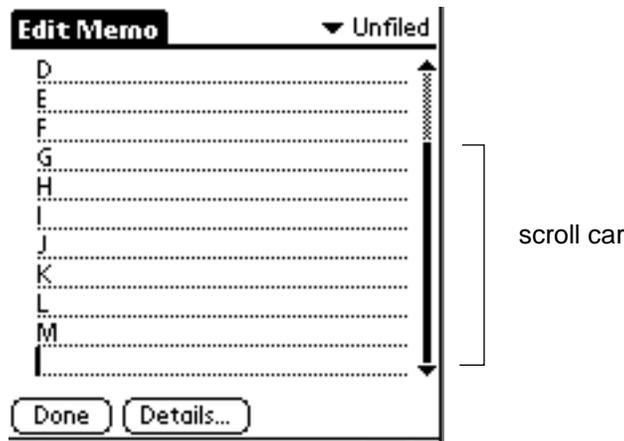
[penUpEvent](#); store x and y coordinates in `EventType`

## Scrollbar Resource

**Overview**

The scroll bar resource helps developers to provide scrolling behavior for fields and tables.

**Example**



**Attributes**

- |                   |   |
|-------------------|---|
| Object Identifier | Name of the object. Assigned by developer and used by Constructor during header file generation and update. |
| Scrollbar ID      | ID of the object (assigned by Constructor).   |
| Left Origin       | Form-relative position of left side of button. Valid values: 0 – 159  |

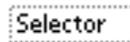
Top Origin	Form-relative position of top of button. Valid values: 0 – 159
Width	Width of the scroll bar in pixels. 7 (the default) is strongly recommended.
Height	Height of the scrollbar in pixels. Valid values: 1 – 160
Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Value	Current top value of the scroll bar's car (movable piece).
Min Value	Position of the scroll car when the scrollbar is at the top. Default should usually be 0.
Max Value	Position of the scroll car when the scrollbar is at the bottom. To compute this value, use the formula: Number of lines – Page size + Overlap.
Page Size	Number of lines to scroll at one time.

## Selector Trigger Resource

**UI Structure**    ControlType

**Overview**    Users can tap a selector trigger to pop up a dialog that lets them select an item. The selected item becomes the label of the selector trigger. For example, a selector trigger for time pops up a time selector. The selected time is entered into the selector trigger.

A selector trigger displays a text label surrounded by a gray rectangular frame, as shown below:



<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer and used by Constructor during header file generation and update.
	Selector Trigger ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of the left side of the object. Valid values: 0 – 159
	Top Origin	Form-relative position of top of object. Valid values: 0 – 159
	Width	Width of the object in pixels. Valid values: 1– 160
	Height	Height of the object in pixels. Height extends two pixels above and one pixel below the 9-point plain font. Height is one pixel above command buttons to accommodate the gray frame. Valid values: 1– 160

Usable	If this box is checked, the object is usable. A nonusable object is not considered part of the application interface and doesn't draw. Nonusable objects can programmatically be set to usable.
Anchor Left	Controls how the object resizes itself when its text label is changed. If checked, the left bound of the object is fixed, if unchecked, the right bound is fixed. Valid values: checked (left bound fixed) unchecked (right bound fixed)
Font	Font used to draw the text label of the button. Choose from the pop-up menu to select one of the fonts.
Label	Text in the selector trigger.

## Event Flow for Selector Trigger Resource

User Action	System Response
Pen goes down on a selector trigger.	<a href="#">penDownEvent</a> ; store x and y coordinates in <code>EventType</code> . <a href="#">ctlEnterEvent</a> ; store selector trigger ID number in <code>EventType</code> .
Pen is lifted from the selector trigger.	<a href="#">ctlSelectEvent</a> ; store selector trigger ID number in <code>EventType</code> . A <code>ctlSelectEvent</code> can only be triggered if a <code>ctlEnterEvent</code> with the same selector trigger ID number has just occurred. <a href="#">frmOpenEvent</a> followed by a <a href="#">winExitEvent</a> , control is passed to a form object. When control is passed back to the selector trigger, a <a href="#">winEnterEvent</a> and a <a href="#">penUpEvent</a> occur.

## Table Resource

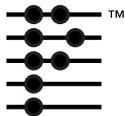
**Overview** The table object allows the developer to organize a collection of objects on the display. For example, a table might contain a column of labels that correspond to a column of fields. Under some circumstances, a one-column table may be preferable to a list.

**Comments** Since tables are scrollable, they may be larger than the display.

### Example



<b>Attributes</b>	Object Identifier	Name of the object. Assigned by developer, used by Constructor during header file generation/update.
	Table ID	ID of the object (assigned by Constructor).
	Left Origin	Form-relative position of left side of the object. Valid values: 0 – 159
	Top Origin	Form-relative position of top of object. Valid values: 0 – 159
	Width	Width of the object in pixels. Valid values: 1– 160
	Height	Height of the object in pixels. Valid values: 1–160
	Rows	Number of rows in the table.
	Columns	Number of columns in the table.
	Column width	Width of the nth column.



# Palm OS User Interface Objects

---

A Palm OS UI object is a C structure that's linked with one or more items on the screen. By changing field values of the C structure, an application can manipulate its user interface. Note that Palm UI objects are just structures, not the more elaborate objects found in some systems. This is useful because a C structure is more compact than other objects could be.

This chapter helps you develop your application's user interface by providing information about each object's structure, associated events, associated UI resource files, and all API calls available for manipulating the structure. It discusses the following objects:

- [Control Objects](#)
- [Date and Time Objects](#)
- [Field Objects](#)
- [Form Objects](#)
- [Insertion Point Object](#)
- [List Object](#)
- [Menu Objects](#)
- [Scrollbar Object](#)
- [Table Objects](#)
- [Window Objects](#)
- [Dynamic User Interface Objects](#)

## A Note on the Rectangle Structure

The `RectangleType` structure is used for describing the area of a rectangle throughout the resources and API. The `RectangleType` defines the top-left corner of a rectangle and its width and height (not the lower-left corner).

## Control Objects

Control objects allow for user interaction when you add them to the forms in your application. There are six types of control objects:

- **Buttons** display a text label in a box. The default style for a button is a text string centered within a rounded rectangle. Buttons have rounded corners unless a rectangular frame is specified. A button without a frame inverts a rounded rectangular region when pressed.

When the user taps a button with the pen, the button highlights until the user releases the pen or drags it outside the bounds of the button.

- A **popup trigger** displays a text label followed by a graphic element (always on the right) that signifies the control initiates a popup list. If the text label changes, the width of the control expands or contracts to the width of the new label plus the graphic element.
- A **selector trigger** displays a text label surrounded by a gray rectangular frame. If the text label changes, the width of the control expands or contracts to the width of the new label.
- A **repeat control** looks like a button. In contrast to buttons, however, users can repeatedly select repeat controls if they don't lift the pen when the control has been selected. The object is selected repeatedly until the pen is lifted.
- **Push buttons** look like buttons, but the frame always has square corners. Touching a push button with the pen inverts the bounds. If the pen is released within the bounds, the button remains inverted.
- **Check boxes** display a setting, either on (checked) or off (unchecked). Touching a check box with the pen toggles the setting. The check box appears as a square, which contains a check mark if the check box's setting is on. A check box can

have a text label attached to it; selecting the label also toggles the check box.

Push buttons and check boxes can be arranged into exclusive groups; one and only one control in a group can be on at a time.

This section provides the following information about control objects:

- [Control Object Events](#)
- [Structure of a Control](#)
- [Associated Resources](#)
- [Control Functions](#)

## Control Object Events

Control objects generate four types of events: [ctlEnterEvent](#), [ctlExitEvent](#), [ctlRepeatEvent](#), and [ctlSelectEvent](#). All these events are generated by the control event handler [CtlHandleEvent](#). All events posted by the handler contain the ID of the control and a pointer to the control data structure

The following table provides an overview of how `CtlHandleEvent` deals with the different events.

## Palm OS User Interface Objects

### Control Objects

---

When CtlHandleEvent receives...	CtlHandleEvent performs these actions...
<a href="#">penDownEvent</a> ; pen position in the bounds of the control object.	Adds a <code>ctlEnterEvent</code> to the event queue
<code>ctlEnterEvent</code>	Inverts the control and tracks the pen until the pen comes up or until the pen is dragged outside the bounds of the control. <ul style="list-style-type: none"><li>• If the pen comes up in the bounds of the control, a <code>ctlSelectEvent</code> is added to the event queue.</li><li>• If the pen is dragged outside the bounds of the control, the control reverts to its original visual state and a <code>ctlExitEvent</code> is added to the event queue.</li></ul>
<code>ctlEnterEvent</code> for a repeat control	Sends a <code>ctlRepeatEvent</code> . When the repeat control receives a <code>ctlRepeatEvent</code> , it tracks the pen for a period of time and then sends another <code>ctlRepeatEvent</code> if the pen is still within the bounds of the control.
<code>ctlExitEvent</code>	Tracks the pen until the pen comes up or is dragged inside the bounds of the control. <ul style="list-style-type: none"><li>• If the pen is dragged into the control, a <code>ctlEnterEvent</code> is added to the event queue.</li><li>• If the pen is released outside the control, no event is posted.</li></ul>

---

## Structure of a Control

### Listing 4.1 ControlType and Auxiliary Structures

```
typedef struct {
    Word          id;
    RectangleType bounds;
    CharPtr       text;
    ControlAttrType attr;
    ControlStyleType style;
```

```
        FontID          font;
        Byte            group;
    } ControlType;

typedef ControlType* ControlPtr;

typedef struct {
    Byte usable        :1;
    Byte enabled       :1;
    Byte visible       :1;
    Byte on            :1;
    Byte leftAnchor    :1;
    Byte frame         :3;
} ControlAttrType;

enum controlStyles {buttonCtl, pushButtonCtl,
                   checkboxCtl, popupTriggerCtl,
                   selectorTriggerCtl, repeatingButtonCtl};
typedef enum controlStyles ControlStyleType;

enum buttonFrames {noButtonFrame,
                  standardButtonFrame, boldButtonFrame,
                  rectangleButtonFrame};
typedef enum buttonFrames ButtonFrameType;
```

#### Fields of a ControlType Structure

The following table lists the fields of a `ControlType` structure and discusses what they do.

Field	Function
<code>id</code>	Symbolic ID of the control, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
<code>text</code>	Pointer to the control's label. If <code>text</code> is <code>NULL</code> , the control has no label. Only buttons, push buttons, and text boxes have text labels.
<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.
<code>attr</code>	Control attributes. The <code>attr</code> field is a bit field that contains the following members: <ul style="list-style-type: none"><li>• A control that doesn't have the <code>usable</code> attribute set is not considered to be part of the interface of the current application, and it doesn't appear on screen.</li><li>• A control that doesn't have the <code>enable</code> attribute set appears "grayed out," and doesn't respond to the pen. Graying out UI elements is strongly discouraged because it's a poor use of screen real estate. Remove the control object instead.</li><li>• The <code>visible</code> attribute is set and cleared internally when the control is drawn and erased.</li><li>• The <code>leftAnchor</code> attribute is used by controls that expand and shrink their width when their label is changed. If the attribute is set, the left bound of the control is fixed.</li><li>• The <code>frame</code> field specifies the type of frame drawn around the button controls. Only button controls use this attribute; for all other controls, the <code>ControlStyle</code> determines the frame.</li></ul>
<code>style</code>	Style of the control: button, push button, check box, popup trigger, popup selector, or repeating button. (See the <code>ControlStyleType</code> enum listed under <a href="#">Structure of a Control</a> .)

<b>Field</b>	<b>Function</b>
font	Font to use to draw the control's label.
group	Group ID of a push button or a check box that is part of an exclusive group. The control routines don't automatically turn one control off when another is selected. It's up to the application or a higher-level object, like a dialog box, to manage this.

---

## Associated 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)

## Control Functions

The following API calls can be used to manipulate control objects.

- [CtlDrawControl](#)
- [CtlEraseControl](#)
- [CtlGetLabel](#)
- [CtlGetValue](#)
- [CtlHandleEvent](#)
- [CtlHideControl](#)
- [CtlHitControl](#)
- [CtlEnabled](#)
- [CtlSetEnabled](#)
- [CtlSetLabel](#)
- [CtlSetUsable](#)
- [CtlSetValue](#)
- [CtlShowControl](#)

## Date and Time Objects

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.

There is no corresponding UI object.

### Date and Time Functions

Currently defined date and time selection functions are [SelectDay](#) and [SelectTime](#).

## Field Objects

A field object displays one or more lines of editable text, supporting these features:

- Proportional fonts (only one font per field)
- Drag-selection
- Scrolling for multiline fields
- Cut, copy, and paste
- Left and right text justification
- Tab stops
- Editable/noneditable attribute
- Expandable field height (the height of the field expands as more text is entered)
- Underlined text (each line of the field is underlined)
- Maximum character limit (the field stops accepting characters when the maximum is reached)
- Special keys (Graffiti strokes) to support cut, copy, and paste
- Insertion point positioning with pen (the insertion point is positioned by touching the pen between characters)
- Scroll bars

The field object does **not** support overstrike input mode; horizontal scrolling; word selection; character filters (for example, only numeric characters accepted); numeric formatting; or special keys for page up, page down, left word, right word, home, end, left margin, right margin, and backspace.

---

**Note:** Field objects can handle line feeds—\0A—but not carriage returns—\0D. PalmRez translates any carriage returns it finds in any Palm OS resources into line feeds, but doesn't touch static data.

---

This section provides the following information about field objects:

- [Field Object Events](#)
- [Structure of a Field](#)
- [Associated Resources](#)
- [Field Functions](#)

## Field Object Events

Events in field objects are handled by [FldHandleEvent](#). `FldHandleEvent` handles events of type [penDownEvent](#), [fldEnterEvent](#), and [keyDownEvent](#).

The following table provides an overview of how `FldHandleEvent` deals with the different events

---

When <code>FldHandleEvent</code> receives...	<code>FldHandleEvent</code> performs these actions...
<code>penDownEvent</code> ; pen position in the bounds of the field object.	Adds a <code>fldEnterEvent</code> to the event queue.
<code>fldEnterEvent</code>	Sets the insertion point position to the position of the pen and tracks the pen until it is released. Drag-selection and drag-scrolling are supported.

---

## Palm OS User Interface Objects

### Field Objects

---

---

When FldHandleEvent receives...	FldHandleEvent performs these actions...
---------------------------------	--

---

A `keyDownEvent` with a special character:

<code>keyDownEvent</code> with up arrow	Moves insertion point up a line.
<code>keyDownEvent</code> with down arrow	Moves insertion point down a line; the insertion point doesn't move beyond the last line that contains text.
<code>keyDownEvent</code> with left arrow	Moves insertion point one character position to the left. When the left margin is reached, move to the end of the previous line.
<code>keyDownEvent</code> with right arrow	Moves insertion point one character position to the right. When the right margin is reached, move to the start of the next line.
<code>keyDownEvent</code> with cut key	Cuts the current selection to the text clipboard.
<code>keyDownEvent</code> with copy key	Copies the current selection to the text clipboard.
<code>keyDownEvent</code> with paste key	Inserts clipboard text into the field at insertion point.

---

## Structure of a Field

The `FieldType` structure and supporting structures are defined as follows:

### Listing 4.2 FieldType Structure

```
typedef struct {
    Word          id;
    RectangleType rect;
    FieldAttrType attr;
    CharPtr       text;
    VoidHand      textHandle;
```

```
LineInfoPtr    lines;
Word           textLen;
Word           textBlockSize;
Word           maxChars;
Word           selFirstPos;
Word           selLastPos;
Word           insPtXPos;
Word           insPtYPos;
FontID         fontID;
} FieldType;
typedef FieldType* FieldPtr;

typedef struct {
    Word usable      :1;
    Word visible     :1;
    Word editable    :1;
    Word singleLine  :1;
    Word hasFocus    :1;
    Word dynamicSize :1;
    Word insPtVisible :1;
    Word dirty       :1;
    Word underlined  :2;
    Word justification:2;
    Word autoShift   :1;
    Word hasScrollBar :1;
} FieldAttrType;

typedef struct {
    Word start;
    Word length;
} LineInfoType;

typedef LineInfoType* LineInfoPtr;
```

## Palm OS User Interface Objects

### Field Objects

---

#### Fields of a Field Structure

The `field` structure has the following fields:

Field	Function
<code>id</code>	ID value specified by the application developer. 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.
<code>attr</code>	Field object attributes. The <code>attr</code> field is a bit field that contains the following members: <code>usable</code> , <code>visible</code> , <code>editable</code> , <code>singleLine</code> , <code>hasFocus</code> , <code>dynamicSize</code> , <code>insPtVisible</code> , <code>dirty</code> , <code>underlined</code> , <code>justification</code> , <code>autoShift</code> , <code>hasScrollBar</code> , and <code>numeric</code> . (see <a href="#">Field Attributes</a> below)
<code>text</code>	Pointer to the NULL-terminated string that is displayed by the field object.
<code>textHandle</code>	Handle to the stored text.
<code>lines</code>	Pointer to an array of <code>LineInfoType</code> structures. There is one entry in this array for each visible line of the text. The <code>LineInfoType</code> structure contains the character position, in the field's text string, of the first character displayed by a line and the number of characters displayed.
<code>textLen</code>	Current number of characters in the string displayed by the field object; the null-terminator is excluded.
<code>textBlockSize</code>	Allocated size of the memory block that holds the field object's text string.
<code>maxChars</code>	Maximum number of characters the field object accepts.
<code>selFirstPos</code>	Starting character position of the current selection.
<code>selLastPos</code>	Ending character position of the current selection. When <code>selFirstPos</code> equals <code>selLastPos</code> , there is no selection.
<code>insPtXPos</code>	Column position of the insertion point.

<b>Field</b>	<b>Function</b>
<code>insPtYPos</code>	Display line where the insertion point is positioned. The first display line is zero.
<code>fontID</code>	Font ID for the field. See <code>Font.h</code> for more information.

---

### Field Attributes

The `attr` field of the Field UI object can have the following values:

- A field object that doesn't have the `usable` attribute set is not considered part of the current interface of the application, and it doesn't appear on screen.
- The `visible` attribute is set or cleared internally when the field object is drawn or erased.
- A field object that doesn't have its `editable` attribute set doesn't accept Graffiti input or edit commands and the insertion point cannot be positioned with the pen.
- If the `singleLine` attribute is set, the height of the `singleLine` field doesn't expand to accommodate more text.
- The `hasFocus` attribute is set internally when the field has the current focus. The blinking insertion point appears in the field that has the current focus.
- If the `dynamicSize` attribute is set, the height of the field expands as characters are entered into the field.
- If the `insPtVisible` attribute is set, the insertion point is scrolled into view. This attribute is set and cleared internally.
- If a field has its `dirty` attribute set, the user has modified the field.
- If a field has its `underlined` attribute set each line of the field, including blank lines, is underlined.
- The `justification` attribute specifies the text alignment (left or right justification only; center justification is not supported).

### Associated Resources

The [Field Resource](#) (tFLD) represents a field on screen.

## **Field Functions**

The following API calls can be used to manipulate field objects.

- [FldCalcFieldHeight](#)
- [FldCompactText](#)
- [FldCopy](#)
- [FldCut](#)
- [FldDelete](#)
- [FldDirty](#)
- [FldDrawField](#)
- [FldEraseField](#)
- [FldFreeMemory](#)
- [FldGetAttributes](#)
- [FldGetBounds](#)
- [FldGetFont](#)
- [FldGetInsPtPosition](#)
- [FldGetMaxChars](#)
- [FldGetNumberOfBlankLines](#)
- [FldGetScrollValues](#)
- [FldGetScrollPosition](#)
- [FldGetSelection](#)
- [FldGetTextAllocatedSize](#)
- [FldGetTextHandle](#)
- [FldGetTextHeight](#)
- [FldGetTextLength](#)
- [FldGetTextPtr](#)
- [FldGetVisibleLines](#)
- [FldGrabFocus](#)
- [FldHandleEvent](#)
- [FldInsert](#)

- [FldMakeFullyVisible](#)
- [FldPaste](#)
- [FldRecalculateField](#)
- [FldReleaseFocus](#)
- [FldScrollable](#)
- [FldScrollField](#)
- [FldSendChangeNotification](#)
- [FldSendHeightChangeNotification](#)
- [FldSetAttributes](#)
- [FldSetBounds](#)
- [FldSetDirty](#)
- [FldSetFont](#)
- [FldSetInsertionPoint](#)
- [FldSetInsPtPosition](#)
- [FldSetMaxChars](#)
- [FldSetScrollPosition](#)
- [FldSetSelection](#)
- [FldSetText](#)
- [FldSetTextAllocatedSize](#)
- [FldSetTextHandle](#)
- [FldSetTextPtr](#)
- [FldSetUsable](#)
- [FldUndo](#)
- [FldWordWrap](#)

## Form Objects

A form object is used as a container for all other UI objects. A form is a window and everything contained within it.

This section provides the following information about form objects:

- [Form Object Events](#)
- [Structure of a Form](#)
- [Associated Resources](#)
- [Form Functions](#)

### Form Object Events

Events in form objects are handled by the [FrmHandleEvent](#) routine.

The following table provides an overview of how `FrmHandleEvent` deals with the different events.

---

When <code>FrmHandleEvent</code> receives...	<code>FrmHandleEvent</code> performs these actions...
<a href="#">penDownEvent</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, <code>CtlHandleEvent</code> is called. If the pen isn't within the bounds of an object, the event is ignored by the form.
<a href="#">keyDownEvent</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="#">ctlEnterEvent</a>	Checks if the control is in an exclusive control group. If it is, it deselects the currently selected control of the group and passes the event and a pointer to the object the event occurred in to <code>CtlHandleEvent</code> . The object pointer is obtained from the event data.

<b>When FrmHandleEvent receives...</b>	<b>FrmHandleEvent performs these actions...</b>
<a href="#">ctlRepeatEvent</a>	Passes the event and a pointer to the object the event occurred in to the appropriate handler. 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.
popSelectEvent	Sets the label of the popup trigger to the current selection of the popup list.
<a href="#">lstEnterEvent</a> or <a href="#">tblEnterEvent</a>	Passes the event and a pointer to the object the event occurred in to the appropriate handler. The object pointer is obtained from the event data.
<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. The table object is also a container object, which may contain a field object. If <code>TblHandleEvent</code> 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="#">frmUpdateEvent</a>	Redraws the form.
<a href="#">menuEvent</a>	Checks if the menu command is one of the system edit menu commands. The system provides a standard edit menu which contains the commands Undo, Cut, Copy, Paste, Select All, and Keyboard. <code>FrmHandleEvent</code> responds to these commands.

---

## Structure of a Form

The `FormType` structure and supporting structures are defined as follows:

### Listing 4.3 FormType Structure and Supporting Structures

```
typedef struct {
    WindowType      window;
    Word            formId;
    FormAttrType    attr;
    WinHandle       bitsBehindForm;
    FormEventHandlerPtr handler;
    Word            focus;
    Word            defaultButton;
    Word            helpRscId;
    Word            menuRscId;
    Word            numObjects;
    FormObjListType* objects;
} FormType;
typedef FormType * FormPtr;

typedef struct {
    Word usable      :1;
    Word enabled     :1;
    Word visible     :1;
    Word dirty       :1;
    Word saveBehind  :1;
    Word graffitiShift:1;
    Word reserved    :11;
} FormAttrType;

typedef struct {
    FormObjectKind  objectType;
    FormObjectType  object;
} FormObjListType;
```

```
typedef union {
    void *                ptr;
    FieldType*           field;
    ControlType*         control;
    ListType*            list;
    TableType*           table;
    FormBitmapType*     bitmap;
    FormLabelType *     label;
    FormTitleType*      title;
    FormPopupType*      popup;
    FormGraffitiStateType* grfState;
    FormGadgetType*     gadget;
    ScrollBarType        scrollBar;
} FormObjectType;

enum formObjects {
    frmFieldObj,
    frmControlObj,
    frmListObj,
    frmTableObj,
    frmBitmapObj,
    frmLineObj,
    frmFrameObj,
    frmRectangleObj,
    frmLabelObj,
    frmTitleObj,
    frmPopupObj,
    frmGraffitiStateObj,
    frmGadgetObj,
    frmScrollbarObj
};
typedef enum formObjects FormObjectKind;
```

## Palm OS User Interface Objects

### *Structure of a Form*

---

```
typedef struct {
    Word usable :1;
} FormObjAttrType;
```

```
typedef struct {
    FormObjAttrType attr;
    PointType      pos;
    Word           rscID;
} FormBitmapType;
```

```
typedef struct {
    FormObjAttrType attr;
    PointType      point1;
    PointType      point2;
} FormLineType;
```

```
typedef struct {
    Word           id;
    FormObjAttrType attr;
    RectangleType rect;
    Word           frameType;
} FormFrameType;
```

```
typedef struct {
    FormObjAttrType attr;
    RectangleType rect;
} FormRectangleType;
```

```
typedef struct {
    Word           id;
    PointType      pos;
    FormObjAttrType attr;
    FontID         fontID;
    char *         text;
} FormLabelType;
```

```
typedef struct {
    RectangleType rect;
    char *      text;
} FormTitleType;

typedef struct {
    unsigned short controlID;
    unsigned short listID;
} FormPopupType;

typedef struct{
    PointerType  pos;
}FrmGraffitiStateType;

typedef struct{
    Word          id;
    FormObjAttrType attr;
    RectangleType rect;
    VoidPtr      date;
}FormGadgetType;
```

### Fields of Form Objects

The form structure has the following fields:

---

<b>Field</b>	<b>Function</b>
window	Structure of the window object that corresponds to the form.
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.

## Palm OS User Interface Objects

### Structure of a Form

---

Field	Function
<code>attr</code>	<p>Form object attributes. The <code>attr</code> field is a bit field that contains the members <code>usable</code>, <code>enable</code>, <code>visible</code>, <code>dirty</code>, <code>saveBehind</code>, and <code>reserved</code>.</p> <ul style="list-style-type: none"><li>• A form that doesn't have the <code>usable</code> attribute set is not considered part of the current interface of the application, and it doesn't appear on screen.</li><li>• When the <code>saveBehind</code> attribute is set, the bits behind the form are saved when the form is drawn.</li><li>• When the <code>visible</code> attribute is set or cleared internally when the field object is drawn or erased.</li><li>• When the <code>dirty</code> attribute is set, the form has been modified in any way. Modifications include the changing of a field or check box. Currently, the system doesn't change the form's <code>dirty</code> attribute when elements of the form are changed.</li><li>• The <code>reserved</code> attribute is reserved for system use.</li></ul>
<code>bitsBehind-Form</code>	<p>Used to save all the bits behind the form so the screen can be properly refreshed when the form is closed. Use this attribute for modal forms.</p>
<code>handler</code>	<p>Routine called when the form needs to handle an event, typically set by the application in the <code>ApplicationHandleEvent</code> function.</p>
<code>focus</code>	<p>Index of a field or table object within the form that contains the focus. Any <code>KeyDownEvent</code> is passed to the object that has the focus.</p>
<code>defaultButton</code>	<p>Index of the object defined as the default button. This value is used by the routine <code>FrmDoDialog</code>.</p>
<code>helpRscId</code>	<p>Resource ID number of the help resource. The help resource is a String resource (type <code>tSTR</code>).</p>
<code>MenuRscId</code>	<p>ID number of a menu bar to use if the form is a menu, or zero if the form is not a menu.</p>
<code>numObjects</code>	<p>Number of objects contained within the form.</p>
<code>objects</code>	<p>Pointer to the array of objects contained within the form.</p>

---

## Associated Resource

The [Form Resource](#) (tFRM) is used to represent forms on screen.

## Form Functions

The following API calls can be used to manipulate form objects.

- [FrmAlert](#)
- [FrmCloseAllForms](#)
- [FrmCopyLabel](#)
- [FrmCopyTitle](#)
- [FrmCustomAlert](#)
- [FrmDeleteForm](#)
- [FrmDispatchEvent](#)
- [FrmDoDialog](#)
- [FrmDrawForm](#)
- [FrmEraseForm](#)
- [FrmGetActiveForm](#)
- [FrmGetActiveFormID](#)
- [FrmGetControlGroupSelection](#)
- [FrmGetControlValue](#)
- [FrmGetFirstForm](#)
- [FrmGetFocus](#)
- [FrmGetFormBounds](#)
- [FrmGetFormId](#)
- [FrmGetFormPtr](#)
- [FrmGetGadgetData](#)
- [FrmGetLabel](#)
- [FrmGetNumberOfObjects](#)
- [FrmGetObjectBounds](#)
- [FrmGetObjectId](#)
- [FrmGetObjectIndex](#)
- [FrmGetObjectPosition](#)
- [FrmGetObjectPtr](#)

## Palm OS User Interface Objects

### *Structure of a Form*

---

- [FrmGetObjectType](#)
- [FrmGetTitle](#)
- [FrmGetUserModifiedState](#)
- [FrmGetWindowHandle](#)
- [FrmGotoForm](#)
- [FrmHandleEvent](#)
- [FrmHelp](#)
- [FrmHideObject](#)
- [FrmInitForm](#)
- [FrmPointInTitle](#)
- [FrmPopupForm](#)
- [FrmReturnToForm](#)
- [FrmSaveAllForms](#)
- [FrmSetActiveForm](#)
- [FrmSetCategoryLabel](#)
- [FrmSetControlGroupSelection](#)
- [FrmSaveAllForms](#)
- [FrmSetActiveForm](#)
- [FrmSetCategoryLabel](#)
- [FrmSetControlGroupSelection](#)
- [FrmSetControlValue](#)
- [FrmSetEventHandler](#)
- [FrmSetFocus](#)
- [FrmSetGadgetData](#)
- [FrmSetMenu](#)
- [FrmSetNotUserModified](#)
- [FrmSetObjectBounds](#)
- [FrmSetObjectPosition](#)
- [FrmSetTitle](#)
- [FrmShowObject](#)
- [FrmUpdateScrollers](#)
- [FrmUpdateForm](#)
- [FrmVisible](#)

## Insertion Point Object

The insertion point is a blinking indicator that shows where text is inserted when users write Graffiti characters or paste clipboard text.

In general, an application doesn't need to be concerned with the insertion point; the Palm OS UI manages the insertion point.

### Insertion Point Functions

For custom insertion point behavior, developers can use the following API calls:

- [InsPtEnable](#)
- [InsPtEnabled](#)
- [InsPtGetHeight](#)
- [InsPtGetLocation](#)
- [InsPtSetHeight](#)
- [InsPtSetLocation](#)

## List Object

The list object appears as a vertical list of choices in a box. The current selection of the list is inverted. If there are more choices than can be displayed, the system draws small arrows (scroll indicators) in the right margin next to the first and last visible choice.

When the pen comes down and up on a scroll indicator, the list is scrolled. When the user scrolls down, the last visible item becomes the first visible item if there are enough items to fill the list. If not, the list is scrolled so that the last item of the list appears at the bottom of the list. The reverse is true for scrolling up. Scrolling doesn't change the current selection.

Bringing the pen down on a list item unhighlights the current selection and highlights the item under the pen. Dragging the pen through the list highlights the item under the pen. Dragging the pen above or below the list causes the list to scroll if it contains more choices than are visible.

When the pen is released over an item, that item becomes the current selection. When the pen is dragged outside the list, the item that was highlighted before the [penDownEvent](#) is highlighted again if it's visible. If it's not, no item is highlighted.

This section provides information about list objects by discussing these topics:

- [List Object Events](#)
- [Structure of a List](#)
- [Associated Resources](#)
- [List Functions](#)

## List Object Events

The list object generates two types of event structures: [lstEnterEvent](#) and [lstSelectEvent](#). Both events are generated by the list event-handler function [LstHandleEvent](#).

The following table provides an overview of how `LstHandleEvent` deals with the different events.

---

When <code>LstHandleEvent</code> receives...	<code>LstHandleEvent</code> performs these actions...
<a href="#">penDownEvent</a>	Adds a <code>lstEnterEvent</code> to the event queue if the pen position is within the bounds of the list.
<code>lstEnterEvent</code>	Tracks the pen until it's released. <ul style="list-style-type: none"><li>• If the pen is released on a list choice, a new selection is made (the data structure is modified) and a <code>lstSelectEvent</code> is added to the event queue.</li><li>• If the pen is released outside the list, the selection is unchanged and no event is posted.</li></ul>

---

## Structure of a List

The `ListType` structure and supporting structures are defined as follows:

### Listing 4.4 List Structure

```
typedef struct {
    Word          id;
    RectangleType bounds;
    ListAttrType  attr;
    CharPtr*      itemsText;
    Word          numItems;
    Word          currentItem;
    Word          topItem;
    FontID        font;
    WinHandle     popupWin;
    ListDrawDataFuncPtr drawItemCallback;
} ListType;

typedef struct {
    Word usable      :1;
    Word enabled     :1;
    Word visible     :1;
    Word poppedUp    :1;
    Word hasScrollBar :1;
    Word search      :1;
    Word reserved    :2;
} ListAttrType;
```

### List Object Fields

The list object has the following fields:

---

Field	Function
id	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> .
bounds	Bounds of the list, relative to the window.
attr	List attributes: <ul style="list-style-type: none"><li>• A form that doesn't have the <a href="#">usable</a> attribute set is not considered part of the current interface of the application, and it doesn't appear on screen.</li><li>• If the <a href="#">enable</a> attribute is set, the user can interact with the list.</li><li>• The <a href="#">visible</a> attribute is set or cleared internally when the field object is drawn or erased.</li><li>• If the <a href="#">poppedUp</a> attribute is set, choices are displayed in a popup window. This attribute is set and cleared internally.</li><li>• If <a href="#">hasScrollbar</a> is set, the field has a scroll bar.</li><li>• If <a href="#">search</a> is set, incremental search is enabled.</li></ul>
itemsText	Pointer to an array of pointers to the text of the choices.
numItems	Number of choices in the list.
currentItem	Currently-selected list choice (0 = first choice).
topItem	First choice displayed in the list.
font	ID of the font used to draw all list text strings.
popupWin	Handle of the window created when a list is displayed if the <a href="#">poppedUp</a> attribute is set.
drawItems- Callback	Function used to draw an item in the list. If NULL, the default drawing routine is used instead.

---

```
void ListDrawDataFuncType (UInt itemNum,  
                           RectanglePtr bounds, CharPtr *itemsText)
```

---

## Associated Resources

The [List Resource](#) (tLST), and [Popup Trigger Resource](#) (tPUT) are used together to represent an active list.

## List Functions

The following API calls can be used to manipulate list objects.

- [LstDrawList](#)
- [LstEraseList](#)
- [LstGetNumberOfItems](#)
- [LstGetSelection](#)
- [LstGetSelectionText](#)
- [LstGetVisibleItems](#)
- [LstHandleEvent](#)
- [LstMakeItemVisible](#)
- [LstNewList](#)
- [LstPopupList](#)
- [LstScrollList](#)
- [LstSetDrawFunction](#)
- [LstSetHeight](#)
- [LstSetListChoices](#)
- [LstSetPosition](#)
- [LstSetSelection](#)
- [LstSetTopItem](#)

## Menu Objects

A menu bar is displayed whenever the user taps a menu icon. The menu bar, a horizontal list of menu titles, appears at the top of the screen in its own window, above all application windows. Pressing a menu title highlights the title and “pulls down” the menu below the title.

User actions have the following effect on a menu:

---

<b>When...</b>	<b>Then...</b>
User drags the pen through the menu.	Command under the pen is highlighted.
Pen is released over a menu item.	That item is selected and the menu bar and menu disappear.
Pen is released outside both the menu bar and the menu.	Both menu and menu bar disappear and no selection is made.
Pen is released in a menu title.	Menu bar and Menu remain displayed until a selection is made from the menu.
Pen is tapped outside menu and menu bar.	Both menu and menu bar are dismissed.
User selects a separator with the pen.	Menu is dismissed but no event is posted.

---

A menu has the following features:

- Item separators, which are lines to group menu items.
- Keyboard shortcuts; the shortcut labels are right justified in menu items.
- A menu remembers its last selection; the next time a menu is displayed the prior selection appears highlighted.
- The bits behind the menu bar and the menus are saved and restored by the menu routines.
- When the menu is visible, the insertion point is turned off.

This section provides information about menu objects by discussing these topics:

- [Menu Events](#)
- [Structure of a Menu](#)
- [Associated Resources](#)
- [Menu Functions](#)

## Menu Events

Menu events are handled by the routine [MenuHandleEvent](#), which handles events of type [penDownEvent](#) and [keyDownEvent](#).

When a menu item is chosen, the menu event handler adds a [menuEvent](#) that identifies the chosen item to the event queue.

## Structure of a Menu

The menu structure and supporting structures are defined as follows:

### Listing 4.5 Menu Structure and Supporting Structures

```
typedef struct {
    WinHandle      barWin;
    WinHandle      bitsBehind;
    WinHandle      savedActiveWin;
    WinHandle      bitsBehindStatus;
    MenuBarAttrType attr;
    SWord          curMenu;
    SWord          curItem;
    long           commandTick;
    SWord          numMenus;
    MenuPullDownPtr menus;
} MenuBarType;

typedef MenuBarType * MenuBarPtr;
```

```
typedef struct {
    Word visible          :1;
    Word commandPending  :1;
    Word insPtEnabled    :1;
} MenuBarAttrType;

typedef struct {
    WinHandle      menuWin;
    RectangleType  bounds;
    WinHandle      bitsBehind;
    RectangleType  titleBounds;
    CharPtr        title;
    Word           numItems;
    MenuItemType  *items;
} MenuPullDownType;

typedef MenuPullDownType * MenuPullDownPtr;

typedef struct {
    Word    id;
    char    command;
    CharPtr itemStr;
} MenuItemType;
```

## Palm OS User Interface Objects

### Menu Objects

---

#### Menu Object Fields

The menu object has the following fields:

---

Field	Function
<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>	Stores currently active window behind the menu.
<code>bitsBehind-Status</code>	Stores the bits behind the status message so that when the message display terminates, the bits can be restored.
<code>attr</code>	Menu bar attributes. The <code>attr</code> field is a bit field that contains the members <code>visible</code> , <code>commandPending</code> , and <code>insPtEnabled</code> . <ul style="list-style-type: none"><li>• If <code>visible</code> is set, the menu bar is drawn.</li><li>• If <code>commandPending</code>, the next key is a command.</li><li>• If <code>insPtEnable</code> is set, the insertion point was on when the menu was drawn.</li></ul>
<code>curMenu</code>	Menu number for the currently visible menu. Menus are numbered sequentially, starting with 0. The value is preserved when the menu bar is dismissed. The next time the menu is displayed, the previously visible pull-down menu can also be redisplayed. A value of -1 indicates that there is no current pull-down menu.
<code>curItem</code>	Item number of the currently highlighted menu item. The items in each menu are numbered sequentially, starting with zero.
<code>commandTick</code>	Stores the tick count at which the status message should be erased.
<code>numMenus</code>	Number of pull-down menus on the menu bar.
<code>menus</code>	Array of <code>MenuPullDownType</code> structures. The <code>MenuPullDownType</code> structure defines a pull-down menu.

---

### Menu Pull-Down Fields

The menu pulldown object has the following fields:

---

<b>Field</b>	<b>Function</b>
menuWin	Handle for the window that contains the menu.
bounds	Position and size, in pixels, of the pull-down menu.
bitsBehind	Handle of a window that contains the region obscured by the menu.
title	Pointer to the menu title (null-terminated).
titleBounds	Bounds of the title in the menubar.
numItems	Number of items in a menu. Separators count as items.
items	Array of <code>MenuItemType</code> structures. A <code>MenuItemType</code> structure defines a menu item.

---

### Menu Item Fields

The menu item object has the following fields:

---

<b>Field</b>	<b>Function</b>
id	ID value specified by the application developer. This ID value is included as part of the event data of a <code>menuEvent</code> .
command	Shortcut key. If you provide shortcuts, make sure that each shortcut is unique among all commands available at that time.
itemStr	Pointer to the text display for a menu item. The shortcut key description is included in this string. The item label and the shortcut key description are delimited with a tab character.

---

#### Associated Resources

The resources MBAR (menu bar) and MENU (menu) are used jointly to represent a menu object on screen.

#### Menu Functions

The following API calls can be used to manipulate menu objects.

- [MenuDispose](#)
- [MenuDrawMenu](#)
- [MenuEraseStatus](#)
- [MenuGetActiveMenu](#)
- [MenuHandleEvent](#)
- [MenuInit](#)
- [MenuSetActiveMenu](#)

## Scrollbar Object

Palm OS 2.0 and later provides vertical scrollbar support. As a result, developers can include scroll bars in forms or tables and the system sends the appropriate events when the end-user interacts with the scroll bar.

Here's what you have to do to include a scroll bar in your GUI:

1. **Create a scroll bar (tSCL) UI resource.**

Provide the ID, the bounds for the scroll bar rectangle. The height has to match the object you want to attach it to (normally a text field). The width should be 7.
2. **Provide a minimum, and maximum value as well as a page size.**
  - minimum is usually 0
  - maximum is usually 0 and set programmatically
  - the page size determines by how many lines the system moves when the text scrolls.

**3. Make the scroll bar part of the form (for tables, place the scroll bar next to the table field programmatically.)**

When you compile your application, the system creates the appropriate scroll bar UI object (see [Scroll Bar UI Object](#))

There are two ways in which the scroll bar and the field (or table field) that it's attached to need to interact:

- When the user adds or removes text, the scroll bar needs to know about the change in size.  
To get this functionality, call `TableHasScrollBar` programmatically. The table or field will then send events whenever the size changes. Your application can catch the events and process them appropriately.
- When the user moves the scroll bar, the text needs to move accordingly. This can either happen dynamically or statically (i.e. after the user has released the scroll bar)  
As a rule, the scroll bar appears on screen as part of the form and is updated appropriately by the system. Applications therefore rarely have to call [SclDrawScrollBar](#), [SclGetScrollBar](#), or [SclSetScrollBar](#). The application usually does call [SclSetScrollBar](#) at initialization time to set the initial position of the scroll bar.
- The system sends the following scroll bar events:
  - [sclEnterEvent](#) is sent when a `penDownEvent` occurs within the bounds of the scroll bar.
  - [sclRepeatEvent](#) is sent when the user drags the scroll bar.
  - [sclExitEvent](#) is sent when the user lifts the pen. This event is sent regardless of previous `sclRepeatEvents`.

Applications that want to support immediate-mode scrolling (that is, scrolling happens as the user drags the pen) need to watch for occurrences of `sclRepeatEvent`.

Application that don't support immediate-mode scrolling should ignore occurrences of `sclRepeatEvent` and wait only for the `sclExitEvent`.

**Listing 4.6 Scroll Bar UI Object**

```
typedef struct {  
    Word usable:          1;  
};
```

## Palm OS User Interface Objects

### Scrollbar Object

---

```
Word visible:      1;
Word highlighted:  1;
Word shown:       1;
Word activeRegion: 4;
} ScrollBarAttrType;
```

```
typedef struct {
    RectangleType bounds;
    Word id;
    ScrollBarAttrType attr;
    Short value;
    Short minValue;
    Short maxValue;
    Short pageSize;
    Short penPosInCar;
    Short savePos;
} ScrollBarType;
```

```
typedef ScrollBarType * ScrollBarPtr;
```

#### Scrollbar Fields

The scrollbar object has the following fields:

---

Field	Function
bounds	Bounds of the scrollbar
id	Developer-defined ID of the scrollbar.

---

<b>Field</b>	<b>Function</b>
<code>attr</code>	Attributes of the scrollbar. <ul style="list-style-type: none"><li>• When the <code>usable</code> attribute is set, the scrollbar is part of the UI.</li><li>• When the <code>visible</code> attribute is set, the scrollbar is visible on screen.</li><li>• When the <code>highlighted</code> attribute is set, the scrollbar is highlighted.</li><li>• The <code>shown</code> attribute has to be true if the scrollbar is visible and if <code>maxValue &gt; minValue</code>.</li><li>• The <code>activeRegion</code> attribute indicates the active region of the scrollbar.</li></ul>
<code>value</code>	Current value of the scroll bar.
<code>minValue</code>	Minimum value (default should be zero).
<code>maxValue</code>	Maximum value. With the <code>scroll car</code> being the dark region in the scrollbar that indicates the position in the document and <code>overlap</code> the number of lines from the bottom of one page to be visible at the top of the next page, this value is usually computed as follows:  number of lines – (page size + overlap)  For example, if you have 100 lines, the scroll car is at maximum at line 90 or 91.
<code>pageSize</code>	Number of lines to scroll when user scrolls one page.
<code>penPosInChar</code>	Used internally.
<code>savePos</code>	Used internally.

---

## Table Objects

The table object is used to organize several types of UI objects. The number of rows and the number of columns must be specified for each table object. A UI object can be placed inside a cell of a table. Tables often consist of rows or columns of the same object. For example, a table might have one column of labels and another column of fields. Tables can only be scrolled vertically. Tables can't include bitmaps.

This section provides information about table objects by discussing these topics:

- [Table Event](#)
- [Structure of a Table](#)
- [Associated Resource](#)
- [Table Functions](#)

### Table Event

The table object generates the event [tblSelectEvent](#). This event contains:

- The table's ID number
- The row of the selected table
- The column of the selected table

When [tblSelectEvent](#) is sent to a table, the table generates an event to handle any possible events within the item's UI object.

### Structure of a Table

The table structure and supporting structures are defined as follows:

```
typedef struct {
    Word                id;
    RectangleType       bounds;
    TableAttrType       attr;
    Word                numColumns;
    Word                numRows;
```

```
    Word                currentRow;
    Word                currentColumn;
    Word                topRow;
    TableColumnAttrType * columnAttrs;
    TableRowAttrType *  rowAttrs;
    TableItemPtr        items;
    FieldType           currentField;
} TableType;
typedef TableType * TablePtr;

typedef struct {
    Word        visible:1;
    Word        editable:1;
    Word        editing:1;
    Word        selected:1;
    Word        hasScrollBar:1.
} TableAttrType;

typedef struct {
    TableItemStyleType  itemType;
    FontID              fontID; //font for drawing text
    Word                intValue;
    CharPtr             ptr;
} TableItemType;
typedef TableItemType * TableItemPtr;

typedef struct {
    Word                width;           // in
pixels
    Boolean             usable;
    Word                spacing;
    TableDrawItemFuncPtr drawCallback;
    TableLoadDataFuncPtr loadDataCallback;
    TableSaveDataFuncPtr saveDataCallback;
} TableColumnAttrType;
```

## Palm OS User Interface Objects

### Table Objects

---

```
typedef struct {
    Word          id;
    Word          height; // row height in pixels
    DWord         data;
    Word          usable;
    Word          selectable;
    Word          invalid; // true if redraw needed
} TableRowAttrType;
```

#### Fields of a Table Structure

The table structure has the following fields:

---

Field	Function
<code>id</code>	ID value specified by the application developer.
<code>bounds</code>	Position and size of the table object.
<code>attr</code>	Table object's attributes. The <code>attr</code> field is a bit field that contains the following members: <ul style="list-style-type: none"><li>• If <code>visible</code> is set, the table is drawn on screen.</li><li>• If <code>editable</code> is set, the user can modify the table.</li><li>• If <code>editing</code> is set, the table is in edit mode.</li><li>• If <code>selected</code> is set, the current item is selected.</li><li>• If <code>hasScrollbar</code> is set, the table has a scroll bar. Note that this attribute can only be set programmatically.</li></ul>
<code>numColumns</code>	Number of columns in the table object.
<code>numRows</code>	Number of rows in the table object.
<code>currentRow</code>	Row of the table set to current.
<code>currentColumn</code>	Column of the table set to current.
<code>topRow</code>	First row in the table object.

---

<b>Field</b>	<b>Function</b>
columnAttrs	Column attributes, such as its width, its usability, and how the column draws itself.
rowAttrs	Row's attributes, such as its ID, height, and whether or not it is usable, selectable, or invalid.
items	Item attributes, such as the item type, font ID, an integer value, and a character pointer.
currentField	Field object the user is currently editing.

---

## Associated Resource

The [Table Resource](#) (tTBL) represents a table on screen.

## Table Functions

The following API calls can be used to manipulate table objects.

- [TblDrawTable](#)
- [TblEditing](#)
- [TblEraseTable](#)
- [TblFindRowData](#)
- [TblFindRowID](#)
- [TblGetBounds](#)
- [TblGetColumnSpacing](#)
- [TblGetColumnWidth](#)
- [TblGetCurrentField](#)
- [TblGetItemBounds](#)
- [TblGetItemInt](#)
- [TblGetLastUsableRow](#)
- [TblGetNumberOfRows](#)
- [TblGetRowData](#)
- [TblGetRowHeight](#)
- [TblGetRowID](#)

## Palm OS User Interface Objects

### *Table Objects*

---

- [TblGetSelection](#)
- [TblGrabFocus](#)
- [TblHandleEvent](#)
- [TblHasScrollBar](#)
- [TblInsertRow](#)
- [TblMarkRowInvalid](#)
- [TblMarkTableInvalid](#)
- [TblRedrawTable](#)
- [TblReleaseFocus](#)
- [TblRemoveRow](#)
- [TblRowInvalid](#)
- [TblRowSelectable](#)
- [TblRowUsable](#)
- [TblSelectItem](#)
- [TblSetBounds](#)
- [TblSetColumnEditIndicator](#)
- [TblSetColumnSpacing](#)
- [TblSetColumnUsable](#)
- [TblSetColumnWidth](#)
- [TblSetCustomDrawProcedure](#)
- [TblSetItemInt](#)
- [TblSetItemPtr](#)
- [TblSetItemStyle](#)
- [TblSetLoadDataProcedure](#)
- [TblSetRowData](#)
- [TblSetRowHeight](#)
- [TblSetRowID](#)
- [TblSetRowSelectable](#)
- [TblSetRowStaticHeight](#)
- [TblSetRowUsable](#)
- [TblSetSaveDataProcedure](#)
- [TblUnhighlightSelection](#)

## Window Objects

A window defines a drawing region. This region may be on the display or in a memory buffer (an off-screen window). Off-screen windows are useful for saving and restoring regions of the display that are obscured by other UI objects. All forms are windows, but not all windows are forms.

The window object is the portion of the form object that determines how the form's window looks and behaves. A window object contains viewing coordinates of the window and clipping bounds.

This section provides information about windows by discussing these topics:

- [Window Events](#)
- [Structure of a Window](#)
- [Window Functions](#)

No resources are associated with window objects.

### Window Events

When a window becomes active, a [winEnterEvent](#) takes place. When the window is deactivated, a [winExitEvent](#) occurs. The `winEnterEvent` usually follows right after a `winExitEvent`; an old window is deactivated just before a new window is activated.

### Structure of a Window

The `WinType` structure is defined as follows:

```
typedef struct WinTypeStruct {
    Word                displayWidth;
    Word                displayHeight;
    VoidPtr             displayAddr;
    WindowFlagsType    windowFlags;
    RectangleType      windowBounds;
    AbsRectType        clippingBounds;
    PointType          viewOrigin;
```

## Palm OS User Interface Objects

### Window Objects

---

```
    FrameBitsType      frameType;  
    GraphicStatePtr   gstate;  
    struct WinTypeStruct* nextWindow;  
} WindowType;
```

```
typedef WindowType * WinPtr;
```

```
typedef WinPtr WinHandle;
```

#### Fields of a Window Structure

---

Field	Function
displayWidth	Width, in pixels, of the display memory buffer (video RAM) for on-screen windows and the width of a memory buffer for off-screen windows.
displayHeight	Height, in pixels, of the device display.
displayAddr	Pointer to the window's display memory buffer.
windowFlags	Window attributes: <i>format</i> , <i>offscreen</i> , <i>modal</i> , <i>focusable</i> , <i>enabled</i> , <i>visible</i> , <i>dialog</i> , and <i>compressed</i> (see next table).
windowBounds	Bounds of the window.
clipping- Bounds	Bounds for clipping any drawing within the window.
viewOrigin	Window origin point on the display.
frameType	Frame's corner diameter, width of shadow, and width of frame.
gstate	State of the graphic mode, pattern mode, font, and underline mode.
nextWindow	Pointer to the next window in a linked list of windows.

---

Window attributes are defined as follows:

<b>Attribute</b>	<b>Set to 0</b>	<b>Set to 1</b>
format	screen mode	generic mode
offscreen	on screen	off screen
modal	modeless window	modal window
focusable	non-focusable	focusable
enabled	disabled	enabled
visible	invisible	visible
dialog	nondialog	dialog
compressed	uncompressed	compressed

## Window Functions

The following API calls can be used to manipulate window objects.

- [ScrDisplayMode](#)
- [WinAddWindow](#)
- [WinClipRectangle](#)
- [WinCopyRectangle](#)
- [WinCreateOffscreenWindow](#)
- [WinCreateWindow](#)
- [WinDeleteWindow](#)
- [WinDisableWindow](#)
- [WinDisplayToWindowPt](#)
- [WinDrawBitmap](#)
- [WinDrawChars](#)
- [WinDrawGrayLine](#)
- [WinDrawGrayRectangleFrame](#)
- [WinDrawInvertedChars](#)

## Palm OS User Interface Objects

### *Window Objects*

---

- [WinDrawLine](#)
- [WinDrawRectangle](#)
- [WinDrawRectangleFrame](#)
- [WinDrawWindowFrame](#)
- [WinDrawWindowFrame](#)
- [WinEnableWindow](#)
- [WinEraseChars](#)
- [WinEraseLine](#)
- [WinEraseRectangleFrame](#)
- [WinEraseWindow](#)
- [WinFillLine](#)
- [WinFillRectangle](#)
- [WinGetActiveWindow](#)
- [WinGetClip](#)
- [WinGetDisplayExtent](#)
- [WinGetDisplayWindow](#)
- [WinGetDrawWindow](#)
- [WinGetFirstWindow](#)
- [WinGetFramesRectangle](#)
- [WinGetPattern](#)
- [WinGetWindowBounds](#)
- [WinGetWindowExtent](#)
- [WinGetWindowFrameRect](#)
- [WinGetWindowPointer](#)
- [WinInitializeWindow](#)
- [WinInvertChars](#)
- [WinInvertLine](#)
- [WinInvertRectangle](#)
- [WinInvertRectangleFrame](#)
- [WinModal](#)
- [WinRemoveWindow](#)
- [WinResetClip](#)
- [WinRestoreBits](#)

- [WinSaveBits](#)
- [WinScrollRectangle](#)
- [WinSetActiveWindow](#)
- [WinSetClip](#)
- [WinSetDrawWindow](#)
- [WinSetPattern](#)
- [WinSetUnderlineMode](#)
- [WinSetWindowBounds](#)
- [WinValidateHandle](#)
- [WinWindowToDisplayPt](#)

## Dynamic User Interface Objects

Palm OS 3.0 provides functions that can be used to create forms and form elements at runtime. Most applications will never need to change any user interface elements at runtime—the built-in applications don't do so, and the Palm user interface guidelines discourage it. However, some applications, such as forms packages, must create their displays at runtime—it is for applications such as these that the Dynamic UI API is provided. If you're not absolutely sure that you need to change your UI dynamically, don't do it—unexpected changes to an application's interface are likely to confuse or frustrate the end user.

Dynamic user interface objects are subject to the following limitations:

- You cannot create tables or Graffiti Shift indicators.
- You cannot create buttons (or repeating buttons) having frames or non-bold frames.
- You cannot move user interface objects after they have been created.

You can use the [FrmNewForm](#) function to create new forms dynamically. Palm's UI guidelines encourage you to keep popup dialogs at the bottom of the screen, using the entire screen width. This isn't enforced by the routine, but is encouraged strongly in order to maintain a look and feel that is consistent with the built-in applications.

## Palm OS User Interface Objects

### *Dynamic User Interface Objects*

---

The [FrmNewLabel](#), [FrmNewBitmap](#), [FrmNewGadget](#), [LstNewList](#), [FldNewField](#) and [CtlNewControl](#) functions can be used to create new objects on forms.

It is fine to add new items to an active form, but doing so is very likely to move the form structure in memory; therefore, any pointers to the form or to controls on the form might change. Make sure to update any variables or pointers that you are using so that they refer to the form's new memory location, which is returned when you create the object.

The [FrmRemoveObject](#) function removes an object from a form. This function doesn't free memory referenced by the object (if any) but it does shrink the form chunk. For best efficiency when removing items from forms, remove items in order of decreasing index values, beginning with the item having the highest index value. When removing items from a form, you need to be mindful of the same concerns as when adding items: the form pointer and pointers to controls on the form may change as a result of any call that moves the form structure in memory.

When creating forms dynamically, or just to make your application more robust, use the [FrmValidatePtr](#) function to ensure that your form pointer is valid and the form it points to is valid. This routine can catch lots of bugs for you - use it!

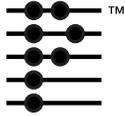
## Dynamic User Interface Functions

Because the functions composing the Dynamic User Interface API are implemented by a variety of distinct objects and managers within Palm OS 3.0, the reference description for a particular function in this API is found in the section of this book dedicated to the manager or object that provides it. For convenience, the list immediately following summarizes the Dynamic User Interface API as a single conceptual entity.

- [CtlNewControl](#)
- [CtlValidatePointer](#)
- [FldNewField](#)
- [FrmNewBitmap](#)

- [FrmNewForm](#)
- [FrmNewGadget](#)
- [FrmNewLabel](#)
- [FrmRemoveObject](#)
- [FrmValidatePtr](#)
- [LstNewList](#)
- [WinValidateHandle](#)





# Using Palm OS UI Managers

---

In contrast to desktop computer operating systems, Palm OS consists of only one library. This library, however, contains several managers, which are groups of functions that work together to implement certain functionality. As a rule, all functions that belong to one manager use the same three-letter prefix and work together to implement a certain aspect of functionality.

In this chapter, you learn about all Palm OS managers that aren't directly responsible for memory management or system management. As you investigate managers more closely you'll find that some of them are mostly services provided by the system, while others contain a large number of API calls.

The managers are presented in alphabetical order for easy access.

- [The Alert Manager](#) lets applications implement modal dialog boxes that display an alert dialog or prompt the user for a response to a question.
- [The Graffiti Manager](#) provides an interface to the Graffiti recognizer. The recognizer converts pen strokes into key events, which are then fed to an application through the event manager.

Most applications never need to call the Graffiti manager directly because the event manager calls it automatically whenever it detects pen strokes in the Graffiti area of the digitizer.

- [The Key Manager](#) provides an interface to the hardware buttons on the Palm OS device. It converts hardware button presses into key events and implements auto-repeat of the buttons.

Most applications never need to call the key manager directly except to change the key repeat rate or poll the current state of the keys.

- [The Pen Manager](#) provides an interface to the digitizer hardware and converts input from the digitizer into pen coordinates.

Most applications never need to call the pen manager directly because any pen activity is automatically returned to the application in the form of events.

- [The Progress Manager](#) provides a mechanism to display changing progress information to the user in a progress dialog. This is useful during any lengthy process such as data transfer during a communications session.

## The Alert Manager

The alert manager provides a simple way for an application to implement modal dialog boxes that display an alert message or prompt the user for a response to a question.

Given a resource ID that defines an alert, the alert manager creates and displays a modal dialog box. When the user taps one of the buttons in the dialog, the alert manager disposes of the dialog box and returns to the caller the item number of the button the user tapped.

There are four types of system-defined alerts:

- Question
- Warning
- Notification
- Error

The alert type determines which icon is drawn in the alert window and which sound plays when the alert is displayed.

### Alert Resource Information

When the alert manager is invoked, it's passed an alert resource (see [Alerts](#)) that contains the following information:

- The rectangle that specifies the size and position of the alert window.
- The alert type (question, warning, notification, or error).
- The null-terminated text string; that is, the message the alert displays.
- The text labels for one or more buttons.

## Alert Manager Functions

The following alert manager functions are available for application use:

- [FrmAlert](#)
- [FrmCustomAlert](#)

## The Graffiti Manager

The Graffiti manager provides an API to the Palm OS Graffiti recognizer. The recognizer converts pen strokes into key events, which are then fed to an application through the event manager.

Most applications never need to call the Graffiti manager directly because it's automatically called by the event manager whenever it detects pen strokes in the Graffiti area of the digitizer.

Special-purpose applications, such as a Graffiti tutorial, may want to call the Graffiti manager directly to recognize strokes in other areas of the screen or to customize the Graffiti behavior.

## Using GrfProcessStroke

[GrfProcessStroke](#) is a high-level Graffiti manager call used by the event manager for converting pen strokes into key events. The call

- Removes pen points from the pen queue
- Recognizes the stroke
- Puts one or more key events into the key queue

`GrfProcessStroke` automatically handles Graffiti ShortCuts and calls the user interface as appropriate to display shift indicators in the current window.

An application can call `GrfProcessStroke` when it receives a [penUpEvent](#) from the event manager if it wants to recognize strokes entered into its application area (in addition to the Graffiti area).

## Using Other High-Level Graffiti Manager Calls

Other high-level calls provided by the Graffiti manager include routines for

- Getting and setting the current Graffiti shift state (caps lock on/off, temporary shift state, etc.)
- Notifying Graffiti when the user selects a different field. Graffiti needs to be notified when a field change occurs so that it can cancel out of any partially entered shortcut and clear its temporary shift state if it's showing a potentially accented character.

## Special-Purpose Graffiti Manager Calls

The remainder of Graffiti manager API routines are for special-purpose use. They are basically all the entry points into the Graffiti recognizer engine and are usually called only by [GrfProcessStroke](#). These special-purpose uses include calls to add pen points to the Graffiti recognizer's stroke buffer, to convert the stroke buffer into a Graffiti glyph ID, and to map a glyph into a string of one or more key strokes.

## Accessing Graffiti ShortCuts

Other routines provide access to the Graffiti ShortCuts database. This is a separate database owned and maintained by the Graffiti manager that contains all of the shortcuts. This database is opened by the Graffiti manager when it initializes and stays open even after applications quit.

The only way to modify this database is through the Graffiti manager API. It provides calls for getting a list of all shortcuts, and for adding, editing, and removing shortcuts. The ShortCuts screen of the Preferences application provides a user-interface for modifying this database.

## Note on Auto Shifting

The Palm OS 2.0 and later automatically uses an upper-case letter under the following conditions:

- Period and space or Return.
- Other sentence terminator (such as ? or !) and space

This functionality requires no changes by the developer, but should be welcome to the end user.

## Note on Graffiti Help

In Palm OS 2.0 and later, applications can pop up Graffiti help by calling `SysGraffitiReferenceDialog` or by putting a special character—`graffitiReferenceChr` from `Chars.h`—on the queue.

Graffiti help is also available through the system Edit menu. As a result, any application that includes the system Edit menu allows users to access Graffiti Help that way. See: `System:HSUtil.h`, `System:HTALSPI.h`, `System:SysConfig.Prv.h`.

## Graffiti Manager Functions

The following functions are available for application use.

- [GrfProcessStroke](#)
- [GrfGetState](#)
- [GrfSetState](#)
- [GrfFlushPoints](#)
- [GrfAddPoint](#)
- [GrfInitState](#)
- [GrfCleanState](#)

- [GrfMatch](#)
- [GrfGetMacro](#)
- [GrfGetAndExpandMacro](#)
- [GrfFilterPoints](#)
- [GrfGetNumPoints](#)
- [GrfGetPoint](#)
- [GrfFindBranch](#)
- [GrfMatchGlyph](#)
- [GrfGetGlyphMapping](#)
- [GrfGetMacroName](#)
- [GrfDeleteMacro](#)
- [GrfAddMacro](#)

## The Key Manager

The key manager manages the hardware buttons on the Palm OS device. It converts hardware button presses into key events and implements auto-repeat of the buttons. Most applications never need to call the key manager directly except to change the key repeat rate or to poll the current state of the keys.

The event manager is the main interface to the keys; it returns a [keyDownEvent](#) to an application whenever a button is pressed. Normally, applications are notified of key presses through the event manager. Whenever a hardware button is pressed, the application receives an event through the event manager with the appropriate key code stored in the event record. The state of the hardware buttons can also be queried by applications at any time through the [KeyCurrentState](#) function call.

The `KeyRates` call changes the auto-repeat rate of the hardware buttons. This might be useful to game applications that want to use the hardware buttons for control. The current key repeat rates are stored in the key manager globals and should be restored before the application exits.

The following functions are available for application use.

- [KeyRates](#)
- [KeyCurrentState](#)

## The Pen Manager

The pen manager manages the digitizer hardware and converts input from the digitizer into pen coordinates. Most applications never need to call the pen manager directly because any pen activity is automatically returned to the application in the form of events.

Pen coordinates are stored in the pen queue as raw, uncalibrated coordinates. When the system event manager routine for removing pen coordinates from the pen queue is called, it converts the pen coordinate into screen coordinates before returning.

The Preferences application provides a user interface for calibrating the digitizer. It uses the pen manager API to set up the calibration which is then saved into the Preferences database. The pen manager assumes that the digitizer is linear in both the x and y directions; the calibration is therefore a simple matter of adding an offset and scaling the x and y coordinates appropriately.

The following functions are available for application use.

- [PenResetCalibration](#)
- [PenCalibrate](#)

## The Progress Manager

The progress manager provides a mechanism to display changing progress information to the user during any lengthy process such as data transfer during a communications session.

You display the progress dialog by calling [PrgStartDialog](#). Then, as your process progresses, you call [PrgUpdateDialog](#) to update the dialog with new information for the user. In your event loop you call [PrgHandleEvent](#) to handle the progress dialog update events queued by `PrgUpdateDialog`. The `PrgHandleEvent` function makes a callback to a `textCallback` function that you supply, to get the latest progress information.

Note that whatever operation you are doing that is the lengthy process, you do the work inside your normal event loop, not in the callback function. That is, you call `EvtGetEvent` and do work when you get a `nilEvent`. Each time you get a `nilEvent`, do a chunk of work, but be sure to continue to call `EvtGetEvent` frequently (like every half second), so that pen taps and other events get noticed quickly enough.

The dialog can display a few lines of text that are automatically centered and formatted. You can also specify an icon that identifies the operation in progress. The dialog has one optional button that can be a cancel or an OK button. The type of the button is automatically controlled by the progress manager and depends on the current progress state (no error, error, or user canceled operation).

### Progress `textCallback` Function

When you want to update the progress dialog with new information, you call the function `PrgUpdateDialog`. To get the current progress information to display in the progress dialog, `PrgHandleEvent` makes a callback to a function, `textCallback`, that you supplied in your call to [PrgStartDialog](#).

The system passes the `textCallback` 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):

<code>&lt;-- Word stage</code>	Current stage (passed from <code>PrgUpdateDialog</code> ).
<code>&lt;--&gt; CharPtr textP</code>	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 <code>stage</code> field. Also, you should append the additional text in the <code>message</code> 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 <code>textLen</code> .
<code>&lt;-- Word textLen</code>	Maximum length of the text buffer <code>textP</code> . Note that this value is set for you by the caller. Be careful not to exceed this length in <code>textP</code> .
<code>&lt;-- CharPtr message</code>	Additional text to display in the dialog (from the <code>messageP</code> parameter to <code>PrgUpdateDialog</code> ). This should be no longer than <code>progressMaxMessage</code> (128)
<code>&lt;-- Err error</code>	Current error (passed from the <code>err</code> parameter to <code>PrgUpdateDialog</code> ).
<code>--&gt; Word bitmapId</code>	Resource ID of the bitmap to display in the progress dialog, if any.
<code>&lt;-- Word canceled</code>	TRUE if user has pressed the cancel button.
<code>&lt;-- Word showDetails</code>	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.)

## Using Palm OS UI Managers

### *The Progress Manager*

---

--> Word textChanged	If TRUE, then update text (defaults to TRUE). You can set this to FALSE to avoid an update to the text.
<-- Word timedOut	TRUE if update caused by a timeout.
<--> ULong 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.

Your `textCallback` function should return a Boolean. Return 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.)

In the `textCallback` 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:** The `textCallback` 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 Function Summary

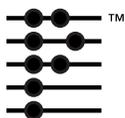
The following functions are available for application use.

- [PrgHandleEvent](#)
- [PrgStartDialog](#)
- [PrgStopDialog](#)
- [PrgUpdateDialog](#)
- [PrgUserCancel](#)

## Using Palm OS UI Managers

### *The Progress Manager*

---



# Palm OS Events

---

Palm OS events are structures that the system passes to the application when the user interacts with the graphical user interface. [How Events Control an Application](#) discusses in detail how this works. This chapter only provides reference-style information about each event. 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="#">keyDownEvent</a>	N.A.
<a href="#">lstEnterEvent</a> , <a href="#">lstExitEvent</a> , <a href="#">lstSelectEvent</a>	List
<a href="#">menuEvent</a>	Menu
<a href="#">nilEvent</a>	N.A.
<a href="#">penDownEvent</a> , <a href="#">penMoveEvent</a> , <a href="#">penUpEvent</a>	N.A. (pen)
<a href="#">popSelectEvent</a>	Popup (Control)
<a href="#">sclEnterEvent</a> , <a href="#">sclRepeatEvent</a> , <a href="#">sclExitEvent</a>	Scroll bar

## Palm OS Events

### *appStopEvent*

---

Event	UI Object
<a href="#">tblEnterEvent</a> , <a href="#">tblExitEvent</a> , <a href="#">tblSelectEvent</a>	Table
<a href="#">winEnterEvent</a> , <a href="#">winExitEvent</a>	Window

---

## 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.

The following data is passed with the event:

`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 following data is passed with the event:

<code>controlID</code>	Developer-defined ID of the control.
<code>pControl</code>	Pointer to a control structure ( <code>ControlType</code> ).

## ctlRepeatEvent

The control routine [CtlHandleEvent](#) sends this event. When `CtlHandleEvent` receives a [ctlEnterEvent](#) in a Repeat control (tREP), it sends a `ctlRepeatEvent`. When `CtlHandleEvent` receives a `ctlRepeatEvent` in a repeat control, it sends another `ctlRepeatEvent` if the pen remains down within the bounds of the control for 1/2 second beyond the last `ctlRepeatEvent`.

The following data is passed with the event:

<code>controlID</code>	Developer-defined ID of the control.
<code>pControl</code>	Pointer to a control structure ( <code>ControlType</code> ).
<code>time</code>	System-ticks count when the event is added to the queue.

## 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.

The following data is passed with the event:

<code>controlID</code>	Developer-defined ID of the control.
------------------------	--------------------------------------

## Palm OS Events

### *daySelectEvent*

---

<code>pControl</code>	Pointer to a control structure ( <code>ControlType</code> ).
<code>on</code>	TRUE when the control is depressed; otherwise, FALSE.

## 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.

The following data is passed with the event:

<code>pSelector</code>	Pointer to a day selector structure ( <code>DaySelectorType</code> ).
<code>selection</code>	Not used.
<code>useThisDate</code>	Set to TRUE to automatically use the selected date.

## fldChangedEvent

The field routine [FldHandleEvent](#) sends this event when the text of a field has been scrolled as a result of drag-selecting. 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.

The following data is passed with the event:

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

## fldEnterEvent

The field routine [FldHandleEvent](#) sends this event when the field receives a [penDownEvent](#) within the bounds of a field. The following data is passed with the event:

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

## fldHeightChangedEvent

The field routine [FldHandleEvent](#) sends this event. The field API supports a feature that allows a field to dynamically resize its visible height as text is added or removed from it. Functions in the field API send a `fldHeightChangedEvent` to change the height of a field. Applications don't usually send or handle this event.

The following data is passed with the event:

`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.

The following data is passed with the event:

<code>formID</code>	Developer-defined ID of the form.
---------------------	-----------------------------------

## 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.

The following data is passed with the event:

<code>formID</code>	Developer-defined ID of the form.
<code>recordNum</code>	Index of record containing the match string.
<code>matchPos</code>	Position of the match.
<code>matchLen</code>	Length of the matched string.
<code>matchFieldNum</code>	Number of the field the matched string was found in.
<code>matchCustom</code>	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.

The following data is passed with the event:

`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.

The following data is passed with the event:

`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.

## 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.

The following data is passed with the event:

<code>formID</code>	Developer-defined ID of the form.
<code>updateCode</code>	The reason for the update request. <code>FrmEraseForm</code> sets this code to zero. Application developers can define their own <code>updateCode</code> . The <code>updateCode</code> is passed as a parameter to <a href="#">FrmUpdateForm</a> .

## frmTitleEnterEvent

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

The following data is passed with the event:

<code>formID</code>	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.

The following data is passed with the event:

<code>formID</code>	Developer-defined ID of the form.
---------------------	-----------------------------------

## 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.

The following data is passed with the event:

<code>chr</code>	ASCII code of character, or zero if the key is a virtual key code; for example, the Find key.
<code>keyCode</code>	Virtual key code; for example, the Find key.
<code>modifiers</code>	One of the following:
<code>shiftKeyMask</code>	True if Graffiti is in case-shift mode.
<code>capsLockMask</code>	True if Graffiti is in cap-shift mode.
<code>numLockMask</code>	True if Graffiti is in numeric-shift mode.
<code>commandKeyMask</code>	True if the Graffiti glyph was the menu command glyph.
<code>optionKeyMask</code>	Not implemented. Reserved.
<code>controlKeyMask</code>	Not implemented. Reserved.
<code>autoRepeatKeyMask</code>	True if generated due to auto-repeat.
<code>doubleTapKeyMask</code>	Not implemented. Reserved.
<code>poweredOnKeyMask</code>	True if the key press caused the system to be powered on.

## IstEnterEvent

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

The following data is passed with the event:

<code>listID</code>	Developer-defined ID of the list.
<code>pList</code>	Pointer to a list structure ( <code>ListType</code> ).
<code>selection</code>	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.

The following data is passed with the event:

<code>listID</code>	Developer-defined ID of the list.
<code>pList</code>	Pointer to a list structure ( <code>ListType</code> ).

## 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.

The following data is passed with the event:

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

## 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 shortcut followed by an available command; for example, Command-C for copy

The following data is passed with the event:

<code>itemID</code>	Item ID of the selected menu command.
---------------------	---------------------------------------

## 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 happens only if the routine `EvtGetEvent`, discussed in *Developing Palm OS Applications, Part I*, 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`.

## penDownEvent

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

The following data is passed with the event:

<code>penDown</code>	Always TRUE.
<code>screenX</code>	Window-relative position of the pen in pixels (number of pixels from the left bound of the window).
<code>screenY</code>	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:

<code>penDown</code>	Always TRUE.
<code>screenX</code>	Window-relative position of the pen in pixels (number of pixels from the left bound of the window).
<code>screenY</code>	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.

The following data is passed with the event:

<code>penDown</code>	Always false.
<code>screenX</code>	Window-relative position of the pen in pixels (number of pixels from the left bound of the window).
<code>screenY</code>	Window-relative position of the pen in pixels (number of pixels from the top left of the window).
<code>start</code>	Display-relative start point of the stroke.
<code>end</code>	Display-relative end point of the stroke.

## popSelectEvent

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

The following data is passed with the event:

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

The following data is passed with the event:

<code>scrollBarID</code>	Developer-defined ID of the scroll bar resource.
<code>pScrollBar</code>	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.

The following data is passed with the event:

<code>scrollBarID</code>	Developer-defined ID of the scroll bar resource.
<code>pScrollBar</code>	Pointer to the scroll bar structure.
<code>value</code>	Initial position of the scroll bar
<code>newValue</code>	New position of the scroll bar. Given <code>value</code> and <code>newValue</code> , 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).

The following data is passed with the event:

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

The following data is passed with the event:

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

The following data is passed with the event:

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

The following data is passed with the event:

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

The following data is passed with the event:

enter- Window	Pointer to the window we are entering. If the window is a form, this is a pointer to a <code>FormType</code> structure; if not, it's a pointer to a <code>WindowType</code> structure.
exit- Window	Pointer 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, this is a pointer to a <code>FormType</code> structure; if not, it's a pointer to a <code>WindowType</code> structure.

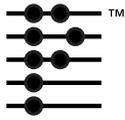
## winExitEvent

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](#)).

The following data is passed with the event:

enterWin- dow	Pointer to the window we are entering. If the window is a form, then this is a pointer to a <code>FormType</code> structure; if not, it's a pointer to a <code>WindowType</code> structure.
exitWindow	Pointer to the window we are exiting. If the window is a form, then this is a pointer to a <code>FormType</code> structure; if not, it's a pointer to a <code>WindowType</code> structure.





# Palm OS User Interface Functions

---

## Category Functions

### CategoryCreateList

**Purpose** Read a database's categories and store them in a list.

**Prototype**

```
void CategoryCreateList (DmOpenRef db,
                        ListPtr listP,
                        Word currentCategory,
                        Boolean showAll,
                        Boolean showUneditables,
                        Byte numUneditableCategories,
                        DWord editingStrID,
                        Boolean resizeList)
```

**Parameters**

<code>db</code>	Opened database containing category info.
<code>listP</code>	List in which to place the categories.
<code>currentCategory</code>	Category to select.
<code>showAll</code>	TRUE to have an All category.
<code>showUneditables</code>	TRUE to show uneditable categories.
<code>numUneditableCategories</code>	Number of categories that the user should not be able to edit. For example, "Unfiled" is an uneditable category in all applications included on the device. Uneditable categories must be kept together in the lowest category numbers.

## Palm OS User Interface Functions

### Category Functions

---

`editingStrID` A resource type to string to edit categories.  
`resizeList` TRUE to resize the list to the number of categories. Set TRUE for popups, FALSE otherwise.

**Result** Returns nothing.

**Comments** Upon return, `listP` points to an allocated list of categories. You must use [CategoryFreeList](#) to free this allocated memory.

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

### CategoryCreateListV10

**Purpose** Read a database's categories and set categories.

**Prototype**

```
void CategoryCreateListV10 ( DmOpenRef db,  
                             ListPtr lst,  
                             Word currentCategory,  
                             Boolean showAll)
```

**Parameters**

<code>db</code>	Database containing categories to extract.
<code>lst</code>	List object to load categories into.
<code>currentCategory</code>	Will be set as the current selection in the resulting list.
<code>showAll</code>	TRUE if an All category should be included in the list.

**Result** Returns nothing.

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

## CategoryEdit

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

**Prototype**

```
Boolean CategoryEdit (const DmOpenRef db,
                      const WordPtr category,
                      const DWord titleStrID,
                      const Byte
numUneditableCategories)
```

**Parameters**

db	Database containing the categories to be edited.
category	Set to the category selected when the dialog is done.
titleStrID	Resource ID of a string resource to display in the title bar of the Edit Categories dialog. If this is 0, the default string (“Edit Categories”) is used.
numUneditableCategories	Number of categories that the user should not be able to edit. For example, “Unfiled” is an uneditable category in all applications included on the device. Uneditable categories must be kept together in the lowest category numbers. Uneditable categories are not displayed in the Edit Categories dialog.

**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.

**Caveat** This function was revised for Palm OS 2.0 and Palm OS 3.0.

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

## **CategoryEditV20**

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

**Prototype** `Boolean CategoryEdit (DmOpenRef db,  
WordPtr categoryP,  
DWord titleStrID)`

**Parameters**

<code>db</code>	Database containing the categories to be edited.
<code>categoryP</code>	Set to the category selected when the dialog is done.
<code>titleStrID</code>	Resource ID of a string resource to display in the title bar of the Edit Categories dialog. If this is 0, the default string (Edit Categories) is used.

**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.

**Caveat** This function was revised for Palm OS 2.0 and Palm OS 3.0.

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

## CategoryEditV10

- Purpose** Event handler for the Edit Categories dialog.
- Prototype** `Boolean CategoryEditV10 (DmOpenRef db,  
WordPtr category)`
- Parameters**
- |                       |  |
|-----------------------|--|
| <code>db</code>       | Database containing the categories to be edited. |
| <code>category</code> | Current 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
- See Also** [CategoryEdit](#), [CategoryEditV20](#)

## CategoryFind

- Purpose** Return the index of the category that matches the name passed.
- Prototype** `Word CategoryFind (DmOpenRef db, CharPtr name)`
- Parameters**
- |                   |   |
|-------------------|---|
| <code>db</code>   | Database to search for the passed category. |
| <code>name</code> | Category name.                              |
- Result** Returns the category index.

## **CategoryFreeList**

**Purpose** This routine unlocks or frees memory locked or allocated by [CategoryCreateList](#).

**Prototype**

```
void CategoryFreeList (DmOpenRef db,  
                      ListPtr listP,  
                      Boolean showAll,  
                      DWord editingStrID)
```

**Parameters**

db	Database containing the categories.
listP	Pointer to the category list.
showAll	TRUE if the list was created with an All category.
editingStrID	A resource type to string to edit categories.

**Comment** Calling this function doesn't remove the categories from the passed database.

**Result** Returns nothing.

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

## CategoryFreeListV10

**Purpose** Unlock or free memory locked or allocated by [CategoryCreateListV10](#) which was attached to the passed List object.

**Prototype** `void CategoryFreeListV10 (DmOpenRef db,  
ListPtr lst)`

**Parameters**

db	Database containing the categories.
lst	Pointer to the category list containing the memory to be freed.

**Result** Returns nothing.

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

## CategoryGetName

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

**Prototype** `void CategoryGetName (DmOpenRef db,  
Word index,  
CharPtr name)`

**Parameters**

db	Database that contains the categories.
index	Category index.
name	Buffer to hold category name. Buffer should be dmCategoryLength in size.

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

## CategoryGetNext

**Purpose** Return the index of the next category, given a category index this routine. Note that categories are not stored sequentially.

**Prototype** `Word CategoryGetNext (DmOpenRef db, Word index)`

**Parameters**

<code>db</code>	Database that contains the categories.
<code>index</code>	Category index.

**Result** Category index of next category.

**Comment** Don't use this function to search for a category. Instead, use it to allow your users to cycle through categories, for example, using the hard-button scroll bars on the device.

**Compatibility Note** In Palm OS 1.0, the system chose `Unfiled` as one category. In Palm OS 2.0 and later, the system skips both `Unfiled` and empty records.

## CategoryInitialize

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

**Prototype** `void CategoryInitialize (  
AppInfoPtr appInfoP,  
Word localizedAppInfoStrID)`

**Parameters**

<code>appInfoP</code>	Application info pointer.
<code>localizedAppInfoStrID</code>	Resource ID of the localized category names

**Result** Returns nothing.

## CategorySelect

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

**Prototype** Boolean CategorySelect (DmOpenRef db,  
FormPtr frm,  
Word ctlID,  
Word lstID,  
Boolean title,  
WordPtr categoryP,  
charPtr categoryName,  
Byte numUneditableCategories,  
DWord editingStrID)

**Parameters**

db	Database that contains the categories.
frm	Form that contains the category popup list.
ctlID	ID of the popup trigger.
lstID	ID of the popup list.
title	TRUE if the popup trigger is on the title line.
categoryP	Current category (index into db structure).
categoryName	Name of the current category.
numUneditableCategories	Number of categories that the user should not be able to edit. Uneditable categories must be kept together in the lowest category number. For the applications included on the device, there is one uneditable category: "Unfiled"
editingStrID	ID of string resource to use for editing categories. If 0, the default, "Edit categories" is used.

**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.

**See Also** [CategorySelectV10](#)

## **CategorySelectV10**

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

**Prototype** `Boolean CategorySelectV10 (DmOpenRef db,  
FormPtr frm,  
Word ctlID,  
Word lstID,  
Boolean title,  
WordPtr categoryP,  
CharPtr categoryName)`

**Parameters**

<code>db</code>	Database that contains the categories.
<code>frm</code>	Form that contains the category popup list.
<code>ctlID</code>	ID of the popup trigger.
<code>lstID</code>	ID of the popup list.
<code>title</code>	TRUE if the popup trigger is on the title line.
<code>categoryP</code>	Current category (index into db structure).
<code>categoryName</code>	Name of the current 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

## CategorySetName

**Purpose** Set the category name and rename bits. A NULL pointer removes the category name.

**Prototype** `void CategorySetName (DmOpenRef db,  
                          Word index,  
                          CharPtr nameP)`

**Parameters**

<code>db</code>	Database containing the categories to change.
<code>index</code>	Index of category to set.
<code>nameP</code>	A category name (null-terminated) or NULL pointer to remove the category.

**Result** Returns nothing.

## CategorySetTriggerLabel

**Purpose** Set the label displayed by the category trigger. The category name is truncated if it's too long.

**Prototype** `void CategorySetTriggerLabel (ControlPtr ctl,  
                                  CharPtr name)`

**Parameters**

<code>ctl</code>	Pointer to control object to relabel.
<code>name</code>	Pointer to the name of the new category.

**Result** Returns nothing.

## CategoryTruncateName

**Purpose** Truncate a category name so that it's short enough to display.

**Prototype** `void CategoryTruncateName (CharPtr name,  
Word maxWidth)`

**Parameters**

<code>name</code>	Category name to truncate.
<code>maxWidth</code>	Maximum size, in pixels, of truncated category (including ellipsis).

**Result** Returns nothing. Stores the changed category in `CharPtr`.

## Clipboard Functions

### ClipboardAddItem

**Purpose** Add the item passed to the specified clipboard. The `format` parameter determines which clipboard (text, ink, etc.) the item is added to.

**Prototype** `void ClipboardAddItem (ClipboardFormatType format,  
VoidPtr ptr,  
Word length)`

**Parameters**

<code>format</code>	Text, ink, bitmap, etc.
<code>ptr</code>	Pointer to the item to place on the clipboard.
<code>length</code>	Size of the item to place on the clipboard.

**Result** Returns nothing.

**See Also** [FldCut](#), [FldCopy](#)

## ClipboardGetItem

- Purpose** Return the handle of the contents of the clipboard of a specified type and the length of a clipboard item.
- Prototype** `VoidHand ClipboardGetItem (`  
`ClipboardFormatType format, WordPtr length)`
- Parameters**
- |                     |  |
|---------------------|--|
| <code>format</code> | Text, ink, bitmap, etc.                      |
| <code>length</code> | Pointer to the length of the clipboard item. |
- Result** Handle of the clipboard item.

## Control Functions

### CtlDrawControl

- Purpose** Draw a control object (and the text in it) on screen. The control is drawn only if its `usable` attribute is `TRUE`.
- Prototype** `void CtlDrawControl (ControlPtr ControlP)`
- Parameters**
- |                       |  |
|-----------------------|--|
| <code>ControlP</code> | Pointer to the control object to draw. |
|-----------------------|--|
- Result** Returns nothing.
- Comments** Sets the `visible` attribute to `TRUE`.
- See Also** [CtlSetUsable](#), [CtlShowControl](#)

#### CtlEnabled

**Purpose** Return TRUE if the control is enabled. Disabled controls do not respond to the pen.

**Prototype** Boolean CtlEnabled (ControlPtr ControlP)

**Parameters** ControlP Pointer to control object.

**Result** Returns TRUE if enabled; FALSE if not.

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

#### CtlEraseControl

**Purpose** Erase a usable and visible control object and its frame from the screen.

**Prototype** void CtlEraseControl (ControlPtr ControlP)

**Parameters** ControlP Pointer to control object to erase.

**Comments** Sets the `visible` attribute to FALSE.

## **CtlGetLabel**

**Purpose** Return a character pointer to a control's text label.

**Prototype** `CharPtr CtlGetLabel (ControlPtr ControlP)`

**Parameters** `ControlP` Pointer to control object.

**Result** Returns a pointer to a null-terminated string.

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

## **CtlGetValue**

**Purpose** Return the current value (on or off) of the specified control. This function is valid only for push buttons and check boxes. The return value is undefined for other control types.

**Prototype** `short CtlGetValue (ControlPtr ControlP)`

**Parameters** `ControlP` Pointer to a control object.

**Result** Returns the current value of the control; 0 = off, 1 = on.

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

## **CtlHandleEvent**

- Purpose** Handle event in the specified control object.
- Prototype** `Boolean CtlHandleEvent (ControlPtr ControlP,  
EventPtr EventP)`
- Parameters**
- |                       |   |
|-----------------------|---|
| <code>ControlP</code> | Pointer to control object.                      |
| <code>EventP</code>   | Pointer to an <code>EventType</code> 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's `usable`, `enabled`, and `visible` attributes must be `TRUE`. This routine handles three type of events: `penDownEvent`, `ctlEnterEvent/ctlRepeatEvent` and `ctlExitEvent`.
- 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 inverted.
- When this routine receives a `ctlEnterEvent` or `ctlRepeatEvent`, it checks that the control ID in the passed event record matches the ID of the specified control. If they match, this routine tracks the pen until it comes up or until it leaves the object's bounds. When that happens, `ctlSelectEvent` is sent to the event queue if the pen came up in the bounds of the control. If the pen exits the bounds, a `ctlExitEvent` is sent to the event queue.

## CtlHideControl

<b>Purpose</b>	Set a control's <code>usable</code> attribute to <code>FALSE</code> and erase the control from the screen. This function calls <a href="#">CtlEraseControl</a> .
<b>Prototype</b>	<code>void CtlHideControl (ControlPtr ControlP)</code>
<b>Parameters</b>	<code>ControlP</code> Pointer to the control object to hide.
<b>Result</b>	Returns nothing.
<b>Comments</b>	A control that is not usable doesn't draw and doesn't respond to the pen. Sets the <code>visible</code> and the <code>usable</code> attributes to <code>FALSE</code> .
<b>See Also</b>	<a href="#">CtlShowControl</a>

## CtlHitControl

<b>Purpose</b>	Simulate tapping a control. This function adds a <a href="#">ctlSelectEvent</a> to the event queue.
<b>Prototype</b>	<code>void CtlHitControl (ControlPtr ControlP)</code>
<b>Parameters</b>	<code>ControlP</code> Pointer to a control object.
<b>Result</b>	Returns nothing.
<b>Comments</b>	Useful for testing.

## **CtlNewControl**

**Purpose** Create a new control object dynamically and install it in the specified form.

**Prototype**

```
ControlPtr CtlNewControl (VoidPtr *formPP,  
                          const Word ID,  
                          const ControlStyleType style,  
                          const CharPtr textP,  
                          const Word x,  
                          const Word y,  
                          const Word width,  
                          const Word height,  
                          const FontID font,  
                          Byte group,  
                          Boolean leftAnchor)
```

<b>Parameters</b>	<--> formPP	Pointer to the pointer to the form in which the new control 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 control, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
	style	A controlStyles value specifying the kind of control to create: button, push button, check box, popup trigger, or popup selector. (You cannot create repeating buttons dynamically.) For more information, see the ControlStyleType enum in <a href="#">Structure of a Control</a> starting on page 130
	textP	Pointer to the control's label text. If textP is NULL, the control has no label. Only buttons, push buttons, and text boxes have text labels. Because the contents of this pointer are copied into their own buffer, you can free the textP

	pointer any time after the <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.
<code>x</code>	Horizontal coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
<code>y</code>	Vertical coordinate of the upper-left corner of the control's boundaries, relative to the window in which it appears.
<code>font</code>	Font used to draw the control's label.
<code>width</code>	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.
<code>height</code>	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.
<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 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.
<code>leftAnchor</code>	<code>TRUE</code> 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.

**See Also** [CtlValidatePointer](#), [FrmRemoveObject](#)

### CtlSetEnabled

**Purpose** Set a control as enabled or disabled. Disabled controls do not respond to the pen.

**Prototype** `void CtlSetEnabled (ControlPtr ControlP,  
Boolean enable)`

**Parameters** `ControlP` Pointer to a control object.  
`enable` TRUE to set enabled; FALSE to set not enabled.

**Result** Returns nothing.

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

### CtlSetLabel

**Purpose** Set the current label for the specified control object. If the control object currently has its `usable` and `visible` attributes set to TRUE, redraw it with the new label.

**Prototype** `void CtlSetLabel (ControlPtr ControlP,  
CharPtr newLabel)`

**Parameters** `ControlP` Pointer to a control object.  
`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.

The pointer passed to this function is stored in the control's data structure; the control doesn't make a copy of the string passed.

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

## CtlSetUsable

<b>Purpose</b>	Set a control usable or not usable by changing the value of its usable attribute.				
<b>Prototype</b>	<pre>void CtlSetUsable (ControlPtr ControlP,                   Boolean usable)</pre>				
<b>Parameters</b>	<table><tr><td>ControlP</td><td>Pointer to a control object.</td></tr><tr><td>usable</td><td>TRUE to set usable; FALSE to set not usable.</td></tr></table>	ControlP	Pointer to a control object.	usable	TRUE to set usable; FALSE to set not usable.
ControlP	Pointer to a control object.				
usable	TRUE to set usable; FALSE to set not usable.				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	Function doesn't usually update the control.				
<b>See Also</b>	<a href="#">CtlEraseControl</a> ,				

## CtlSetValue

<b>Purpose</b>	Set the current value (on or off) of the specified control. If the control is visible, it's visually updated.				
<b>Prototype</b>	<pre>void CtlSetValue (ControlPtr ControlP,                  short newValue)</pre>				
<b>Parameters</b>	<table><tr><td>ControlP</td><td>Pointer to a control object.</td></tr><tr><td>newValue</td><td>0 = off, non-zero = on.</td></tr></table>	ControlP	Pointer to a control object.	newValue	0 = off, non-zero = on.
ControlP	Pointer to a control object.				
newValue	0 = off, non-zero = on.				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	Function works only with push buttons and check boxes. Other controls ignore calls to this function.				
<b>See Also</b>	<a href="#">CtlGetValue</a>				

## **CtlShowControl**

- Purpose** Set a control's usable attribute to TRUE and draw the control on the screen. This function calls [CtlDrawControl](#).
- Prototype** `void CtlShowControl (ControlPtr ControlP)`
- Parameters** `ControlP`            Pointer to a control object.
- 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 Also** [CtlHideControl](#)

## **CtlValidatePointer**

- Purpose** Returns TRUE if the specified pointer references a valid control object.
- Prototype** `Boolean CtlValidatePointer (  
                  const ControlPtr pControl)`
- Parameters** `--> pControl`        Pointer to a control.
- 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.
- See Also** [FrmValidatePtr](#), [WinValidateHandle](#)

## Field UI Functions

### FldCalcFieldHeight

- Purpose** Determine the height of a field for a string.
- Prototype** `Word FldCalcFieldHeight ( CharPtr chars,  
Word maxWidth)`
- Parameters**
- |                       |                                      |
|-----------------------|--------------------------------------|
| <code>chars</code>    | Pointer to a null-terminated string. |
| <code>maxWidth</code> | Maximum line width in pixels.        |
- Result** Returns total number of lines needed to draw the string passed.

### FldCompactText

- Purpose** Compact the memory block that contains the text of the field to release any unused space.
- Prototype** `void FldCompactText (FieldPtr fld)`
- Parameters**
- |                  |   |
|------------------|---|
| <code>fld</code> | Pointer to a field object (FieldType data structure). |
|------------------|---|
- Result** Returns nothing.
- Comments** As characters are added to the text of a field, the block that contains the text is grown. The block is expanded in chunks so that it doesn't have to expand each time a character is added. This expansion results in some unused space in the text block.
- Applications should call this function should 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, when the form is being closed.

## FldCopy

**Purpose** Copy the current selection to the text clipboard.

**Prototype** `void FldCopy (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).

**Result** Returns nothing.

**Comments** This function leaves the current selection highlighted. This functions replaces anything previously in the text clipboard. If there is no selection, this function does nothing.

**See Also** [FldCut](#), [FldPaste](#)

## FldCut

**Purpose** Copy the current selection to the text clipboard, delete the selection from the field, and redraw the field.

**Prototype** `void FldCut (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).

**Result** Returns nothing.

**Comments** Anything previously in the text clipboard is replaced by this function. If there is no selection, this function does nothing.

**See Also** [FldCopy](#), [FldPaste](#)

## FldDelete

**Purpose** Delete the specified range of characters from the field and redraw the field.

**Prototype** `void FldDelete (FieldPtr fld,  
Word start,  
Word end)`

**Parameters**

<code>fld</code>	Pointer to the field object to delete from.
<code>start</code>	Starting character position.
<code>end</code>	Ending character position.

**Result** Returns nothing.

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

## FldDirty

**Purpose** Return TRUE if the field has been modified by the user since the text value was set ([FldSetText](#)).

**Prototype** `Boolean FldDirty (FieldPtr fld)`

**Parameters**

<code>fld</code>	Pointer to a field object (FieldType data structure).
------------------	---

**Result** Returns TRUE if the field has been modified by the user, FALSE if the field has not been modified.

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

## **FldDrawField**

**Purpose** Draw the text of the field. The field's `usable` attribute must be `TRUE` or the field won't be drawn.

**Prototype** `void FldDrawField (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (`FieldType` data structure).

**Result** Returns nothing.

**Comments** 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.

**Prototype** `void FldEraseField (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (`FieldType` data structure).

**Result** Returns nothing.

**Comments** The function doesn't modify the contents of the field. If the field has the focus, the blinking insertion point is turned off.

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

## FldFreeMemory

<b>Purpose</b>	Release the memory allocated to the text of a field and the associated word-wrapping information.
<b>Prototype</b>	<code>void FldFreeFieldMemory (FieldPtr fld)</code>
<b>Parameters</b>	<code>fld</code> Pointer to a field object (FieldType data structure).
<b>Result</b>	Returns nothing.
<b>Comments</b>	This function releases <ul style="list-style-type: none"><li>• The memory allocated to the text of a field—the memory block that the <code>text</code> member of the FieldType data structure points to.</li><li>• The memory allocated to hold the display lines information—the memory block that the <code>lines</code> member of the FieldType data structure points to.</li></ul> This function doesn't affect the display of the field.

## FldGetAttributes

<b>Purpose</b>	Return the attributes of a field.
<b>Prototype</b>	<code>void FldGetAttributes (FieldPtr fld,                           FieldAttrPtr attrP)</code>
<b>Parameters</b>	<code>fld</code> Pointer to a FieldType structure. <code>attrP</code> Pointer to FieldAttrType, see Field.h.
<b>Result</b>	Returns nothing.
<b>See Also</b>	<a href="#">FldSetAttributes</a>

## **FldGetBounds**

**Purpose** Return the current bounds of a field.

**Prototype** `void FldGetBounds (FieldPtr fld,  
RectanglePtr rect)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).  
`rect` Pointer to a RectangleType structure.

**Result** Returns nothing. Stores the field's bounds in the RectangleType structure reference by bounds.

**Comments** Returns the `rect` field of the FieldType structure.

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

## **FldGetFont**

**Purpose** Return the ID of the font used to draw the text of a field.

**Prototype** `FontID FldGetFont (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).

**Result** Returns the ID of the font.

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

## FldGetInsPtPosition

- Purpose** Return the insertion point position within the string.
- Prototype** `Word FldGetInsPtPosition (FieldPtr fld)`
- Parameters** `fld` Pointer to a field object (FieldType data structure).
- Result** Returns the character position of the insertion point.
- Comments** The insertion point is to the left of the character position 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 character position after the line feed character is the beginning of the next line.
- See Also** [FldSetInsPtPosition](#)

## FldGetMaxChars

- Purpose** Return the maximum number of characters the field accepts.
- Prototype** `Word FldGetMaxChars (FieldPtr fld)`
- Parameters** `fld` Pointer to a field object (FieldType data structure).
- Result** Returns the maximum number of characters the user is allowed to enter.
- See Also** [FldSetMaxChars](#)

## **FldGetNumberOfBlankLines**

**Purpose** Return the number of blank lines that are displayed at the bottom of a field. This routine is useful for updating a scroll bar after characters have been removed from the text in a field.

**Prototype** `Word FldGetNumberOfBlankLines (FieldPtr fld)`

**Parameters** `fld` Pointer to a `FieldType` structure.

**Result** Returns the number of blank lines visible.

## **FldGetScrollPosition**

**Purpose** Return the string position of the first character in the first line of a field.

**Prototype** `Word FldGetScrollPosition (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (`FieldType` data structure).

**Result** Returns the character position of the first visible character.

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

## FldGetScrollValues

**Purpose** Return the values necessary to update a scroll bar.

**Prototype**

```
void FldGetScrollValues (FieldPtr fld,  
                        WordPtr scrollPosP,  
                        WordPtr textHeightP,  
                        WordPtr fieldHeightP)
```

**Parameters**

<code>fld</code>	Pointer to a <code>FieldType</code> structure.
<code>scrollPosP</code>	Return scroll position here.
<code>textHeightP</code>	Return text height here.
<code>fieldHeightP</code>	Return field height here.

**Result** Returns nothing. Stores the position, text height, and field height in the parameters passed in.

## **FldGetSelection**

**Purpose** Return the current selection of a field.

**Prototype** `void FldGetSelection (FieldPtr fld,  
WordPtr startPosition,  
WordPtr endPosition)`

**Parameters**

<code>fld</code>	Pointer to a field object (FieldType data structure).
<code>startPosition</code>	Pointer to start-character position of selected range of characters.
<code>endPosition</code>	Pointer to end-character position of selected range of characters.

**Result** Returns the start and end position in `startPosition` and `endPosition`.

**Comments** The first character in a field is at position zero.  
If the user has selected the first five characters of a field, `startPosition` will contain the value 0 and `endPosition` the value 5.

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

## **FldGetTextAllocatedSize**

**Purpose** Return the number of characters allocated to hold the field's text string. Don't confuse this number with the length of the text string.

**Prototype** `Word FldGetTextAllocatedSize (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object.

**Result** Returns the number of characters allocated for the field's text.

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

## **FldGetTextHandle**

**Purpose** Return a handle to the block that contains the text string of a field.

**Prototype** `Handle FldGetTextHandle (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).

**Result** Returns the handle of the text string of a field; 0 is a possible value.

**Comments** If 0 is returned, no handle has been allocated for the field pointer.

**See Also** [FldSetTextHandle](#), [FldGetTextPtr](#)

## **FldGetTextHeight**

**Purpose** Return the number of lines of text that the specified field has.

**Prototype** `Word FldGetTextHeight (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).

**Result** Returns the number of lines with text.

**Comments** Empty lines are not counted.

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

## **FldGetTextLength**

**Purpose** Return the length of the text string of a field object.

**Prototype** `Word FldGetTextLength (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).

**Result** Returns the length of a field's text string.

## FldGetTextPtr

**Purpose** Return a pointer to the text string of a field, or NULL.

**Prototype** `CharPtr FldGetTextPtr (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).

**Result** Returns a pointer to the text string of a field; NULL is a possible value.

**See Also** [FldSetTextPtr](#), [FldGetTextHandle](#)

## FldGetVisibleLines

**Purpose** Return the number of lines that can be displayed within the visible bounds of the field.

**Prototype** `Word FldGetVisibleLines (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).

**Result** Returns the number of lines.

## FldGrabFocus

**Purpose** Turn the insertion point on (if the specified field is visible) and position the blinking insertion point in the field.

**Prototype** `void FldGrabFocus (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).

**Result** Returns nothing.

**Comments** This function sets the field attribute `hasFocus` to TRUE.

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

## FldHandleEvent

<b>Purpose</b>	Handles the following events: <a href="#">keyDownEvent</a> , <a href="#">penDownEvent</a> , and <a href="#">fldEnterEvent</a> . The field's <code>editable</code> and <code>usable</code> attributes must be set to <code>TRUE</code> .	
<b>Prototype</b>	<pre>Boolean FldHandleEvent (FieldPtr fld,                         EventPtr EventP)</pre>	
<b>Parameters</b>	<code>fld</code>	Pointer to a field object ( <code>FieldType</code> data structure).
	<code>EventP</code>	Pointer to an event ( <code>EventType</code> data structure).
<b>Result</b>	Returns <code>TRUE</code> if the event was handled.	
<b>Comments</b>	<p>When a <a href="#">keyDownEvent</a> occurs, 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.</p> <p>When a <a href="#">penDownEvent</a> occurs, an "editable" field sends a <a href="#">fldEnterEvent</a> to the event queue.</p> <p>When a <a href="#">fldEnterEvent</a> occurs, the field grabs the focus and the insertion point is placed in the specified position.</p> <p>If the event alters the contents of the field, this function visually updates the field.</p> <p>This function doesn't handle any events if the field is not editable.</p>	

## **FldInsert**

**Purpose** Replace the current selection with the string passed.

**Prototype** `Boolean FldInsert (FieldPtr fld,  
CharPtr insertChars,  
Word insertLen)`

**Parameters**

<code>fld</code>	Pointer to the field object to insert to.
<code>insertChars</code>	Text string to be inserted.
<code>insertLen</code>	Length of the text string to be inserted.

**Result** Returns `TRUE` if string was successfully inserted; otherwise, `FALSE`.

**Comments** If there is no current selection, the string passed is inserted at the position of the insertion point.

**See Also** [FldPaste](#), [FldDelete](#), [FldCut](#), [FldCopy](#)

## **FldMakeFullyVisible**

**Purpose** Cause a dynamically resizable field to expand its height to make its text fully visible.

**Prototype** `Boolean FldMakeFullyVisible (FieldPtr fld)`

**Parameters**

<code>fld</code>	Pointer to a field object.
------------------	----------------------------

**Result** Returns `TRUE` if the field was not fully visible; `FALSE` otherwise.

**Comments** If the field's height changes, this function sends a [fldHeightChangedEvent](#) via the event queue.

**Caveats** If the field is in a table, the table resizes it; otherwise, it's not resized.

## FldNewField

<b>Purpose</b>	Create a new field object dynamically and install it in the specified form.												
<b>Prototype</b>	<pre>FieldPtr FldNewField (VoidPtr *formPP,                       const Word id,                       const Word x,                       const Word y,                       const Word width,                       const Word height,                       const FontID font,                       DWord maxChars,                       Boolean editable,                       Boolean underlined,                       Boolean singleLine,                       Boolean dynamicSize,                       JustificationType justification,                       Boolean autoShift,                       Boolean hasScrollBar,                       Boolean numeric)</pre>												
<b>Parameters</b>	<table><tr><td style="vertical-align: top;"><code>&lt;--&gt; formPP</code></td><td>Pointer to the pointer to the form in which the new field is installed. This value is not a handle; that is, the old <code>formPP</code> value is not necessarily valid after this function returns. In subsequent calls, always use the new <code>formPP</code> value returned by this function.</td></tr><tr><td style="vertical-align: top;"><code>id</code></td><td>Symbolic ID of the field, specified by the developer. By convention, this ID should match the resource ID (not mandatory).</td></tr><tr><td style="vertical-align: top;"><code>x</code></td><td>Horizontal coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.</td></tr><tr><td style="vertical-align: top;"><code>y</code></td><td>Vertical coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.</td></tr><tr><td style="vertical-align: top;"><code>width</code></td><td>Width of the field, expressed in pixels.</td></tr><tr><td style="vertical-align: top;"><code>height</code></td><td>Height of the field, expressed in pixels.</td></tr></table>	<code>&lt;--&gt; formPP</code>	Pointer to the pointer to the form in which the new field is installed. This value is not a handle; that is, the old <code>formPP</code> value is not necessarily valid after this function returns. In subsequent calls, always use the new <code>formPP</code> value returned by this function.	<code>id</code>	Symbolic ID of the field, specified by the developer. By convention, this ID should match the resource ID (not mandatory).	<code>x</code>	Horizontal coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.	<code>y</code>	Vertical coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.	<code>width</code>	Width of the field, expressed in pixels.	<code>height</code>	Height of the field, expressed in pixels.
<code>&lt;--&gt; formPP</code>	Pointer to the pointer to the form in which the new field is installed. This value is not a handle; that is, the old <code>formPP</code> value is not necessarily valid after this function returns. In subsequent calls, always use the new <code>formPP</code> value returned by this function.												
<code>id</code>	Symbolic ID of the field, specified by the developer. By convention, this ID should match the resource ID (not mandatory).												
<code>x</code>	Horizontal coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.												
<code>y</code>	Vertical coordinate of the upper-left corner of the field's boundaries, relative to the window in which it appears.												
<code>width</code>	Width of the field, expressed in pixels.												
<code>height</code>	Height of the field, expressed in pixels.												

## Palm OS User Interface Functions

### Field UI Functions

---

<code>font</code>	Font to use to draw the field's text.
<code>maxChars</code>	Maximum number of characters held by the field this function creates.
<code>editable</code>	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.
<code>underlined</code>	Pass <code>TRUE</code> to create a field that underlines the text it displays.
<code>singleLine</code>	Pass <code>TRUE</code> to create a field that can display only a single line of text.
<code>dynamicSize</code>	Pass <code>TRUE</code> to create a field that resizes dynamically according to the amount of text it displays.
<code>justification</code>	Pass either of the selectors <code>leftAlign</code> or <code>rightAlign</code> to specify left justification or right justification, respectively. The <code>centerAlign</code> selector is not supported.
<code>autoShift</code>	Pass <code>TRUE</code> to specify the use of Palm OS 2.0 (and later) auto-shift rules. If checked, 2.0 auto-shift rules are applied, causing the system to use an upper-case letter automatically: <ul style="list-style-type: none"><li>• after an empty field</li><li>• after a period or other sentence terminator, such as a question mark ( ? ) or an exclamation point ( ! ).</li><li>• after two spaces.</li></ul>
<code>hasScrollBar</code>	Pass <code>TRUE</code> to attach a scroll bar control to the field this function creates.
<code>numeric</code>	

**Result** Returns a pointer to the new field object.

**See Also** [FrmValidatePtr](#), [WinValidateHandle](#), [CtlValidatePointer](#), [FrmRemoveObject](#)

## FldPaste

- Purpose** Replace the current selection in the field with the contents of the text clipboard.
- Prototype** `void FldPaste (FieldPtr fld)`
- Parameters** `fld` Pointer to a field object (FieldType data structure).
- Result** Returns nothing
- Comments** The function performs these actions:
- Positions the insertion point after the last character inserted.
  - 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,
  - Doesn't delete the current selection if there is no text in the clipboard.
- See Also** [FldInsert](#), [FldDelete](#), [FldCut](#), [FldCopy](#)

## **FldRecalculateField**

**Purpose** Update the structure that contains the word-wrapping information for each visible line.

**Prototype** `void FldRecalculateField (FieldPtr fld,  
Boolean redraw)`

**Parameters**

<code>fld</code>	Pointer to a field object (FieldType data structure).
<code>redraw</code>	If TRUE, redraws the field.

**Result** Returns nothing.

**Comments** If necessary, this function reallocates the memory block that contains the displayed lines information, the block pointed to by the `lines` member in the FieldType data structure.

Call this function if the field's data structure is modified in a way that invalidates the visual appearance of the field.

## **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.

**Prototype** `void FldReleaseFocus (FieldPtr fld)`

**Parameters**

<code>fld</code>	Pointer to a field object (FieldType data structure).
------------------	---

**Result** Returns nothing.

**Comments** This function sets the field attribute `hasFocus` to FALSE.

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

## FldScrollable

- Purpose** Return TRUE if the field is scrollable in the specified direction.
- Prototype** `Boolean FldScrollable (FieldPtr fld,  
DirectionType direction)`
- Parameters**
- |                        |   |
|------------------------|---|
| <code>fld</code>       | Pointer to a field object (FieldType data structure). |
| <code>direction</code> | Either the string “up” or the string “down”.          |
- Result** Returns TRUE if the field is scrollable; FALSE otherwise.
- See Also** [FldScrollField](#)

## FldScrollField

- Purpose** Scroll a field up or down by the number of lines specified.
- Prototype** `void FldScrollField (FieldPtr fld,  
Word linesToScroll,  
DirectionType direction)`
- Parameters**
- |                            |   |
|----------------------------|---|
| <code>fld</code>           | Pointer to a field object (FieldType data structure). |
| <code>linesToScroll</code> | Number of lines to scroll.                            |
| <code>direction</code>     | Either the string “up” or the string “down”.          |
- Result** Returns nothing.
- Comments** This function can’t scroll horizontally, that is, right or left. The field object is redrawn if it’s scrolled.
- See Also** [FldScrollable](#)

## **FldSendChangeNotification**

**Purpose** Send a [fldChangedEvent](#) via the event queue.

**Prototype** `void FldSendChangeNotification (FieldPtr fld)`

**Parameters** `fld` Pointer to a field object.

**Result** Returns nothing.

## **FldSendHeightChangeNotification**

**Purpose** Send a [fldHeightChangedEvent](#) via the event queue.

**Prototype** `void FldSendHeightChangeNotification (  
FieldPtr fld,  
Word pos,  
Short numLines)`

**Parameters** `fld` Pointer to a field object.  
`pos` Character position of the insertion point.  
`numLines` New number of lines in the field.

**Result** Returns nothing.

## FldSetAttributes

- Purpose** Set the attributes of a field.
- Prototype** `void FldSetAttributes (FieldPtr fld,  
FieldAttrPtr attrP)`
- Parameters**
- |                    |  |
|--------------------|--|
| <code>fld</code>   | Pointer to a <code>FieldType</code> structure. |
| <code>attrP</code> | Pointer to the attributes.                     |
- Result** Returns nothing.
- See Also** [FldGetAttributes](#)

## FldSetBounds

- Purpose** Change the position or size of a field.
- Prototype** `void FldSetBounds (FieldPtr fld,  
RectanglePtr rect)`
- Parameters**
- |                   |  |
|-------------------|--|
| <code>fld</code>  | Pointer to a field object ( <code>FieldType</code> data structure).                            |
| <code>rect</code> | Pointer to a <code>RectangleType</code> structure that contains the new bounds of the display. |
- Result** Returns nothing.
- Comments** If the field is visible, the field is redrawn within its new bounds. The memory block that contains the word-wrapping information will be resized if the number of visible lines is changed. The insertion point is assumed to be off when this routine is called.
- Caveats** Don't change the width of the object while it's visible.
- See Also** [FldGetBounds](#)

## **FldSetDirty**

**Purpose** Set whether the field has been modified.

**Prototype** `void FldSetDirty (FieldPtr fld, Boolean dirty)`

**Parameters**

<code>fld</code>	Pointer to a field object.
<code>dirty</code>	TRUE if the text is modified.

**Result** Returns nothing.

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

## **FldSetFont**

**Purpose** Set the font of the field, update the word-wrapping information, and draw the field if the field is visible.

**Prototype** `void FldSetFont (FieldPtr fld, FontID fontID)`

**Parameters**

<code>fld</code>	Pointer to a field object (FieldType data structure).
<code>fontID</code>	ID of new font.

**Result** Returns nothing.

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

## FldSetInsertionPoint

**Purpose** Set the location of the insertion point based on a specified string position. This routine differs from [FldSetInsPtPosition](#) in that it doesn't make the character position visible.

**Prototype** `void FldSetInsertionPoint (FieldPtr fld, Word pos)`

**Parameters**

<code>fld</code>	Pointer to a <code>FieldType</code> structure.
<code>pos</code>	Character position in the text of the field

**Result** Nothing.

**Caution** `FldSetInsertionPoint` doesn't make the field the current focus of input if it was not already.

## FldSetInsPtPosition

**Purpose** Set the location of the insertion point for a given string position.

**Prototype** `void FldSetInsPtPosition (FieldPtr fld, Word pos)`

**Parameters**

<code>fld</code>	Pointer to a field object ( <code>FieldType</code> data structure).
<code>pos</code>	Character position of insertion point.

**Result** Returns nothing.

**Comments** If the position is beyond the visible text, then the insertion point is disabled.

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

## **FldSetMaxChars**

**Purpose** Set the maximum number of characters the field accepts.

**Prototype** `void FldSetMaxChars (FieldPtr fld, Word maxChars)`

**Parameters**

<code>fld</code>	Pointer to a field object (FieldType data structure).
<code>maxChars</code>	Maximum number of characters the user may enter.

**Result** Returns nothing.

**Comments** Line feed characters are counted when the number of characters is determined.

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

## **FldSetScrollPosition**

**Purpose** Set the string position of the first character in the first line of a field. Redraw the field if necessary.

**Prototype** `void FldSetScrollPosition (FieldPtr fld, Word pos)`

**Parameters**

<code>fld</code>	Pointer to a field object (FieldType data structure).
<code>pos</code>	Character position of first visible character.

**Result** Returns nothing.

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

## FldSetSelection

<b>Purpose</b>	Set the current selection in a field and highlight the selection if the field is visible.						
<b>Prototype</b>	<pre>void FldSetSelection (FieldPtr fld,                     Word startPosition,                     Word endPosition)</pre>						
<b>Parameters</b>	<table><tr><td><code>fld</code></td><td>Pointer to a field object (FieldType data structure)</td></tr><tr><td><code>startPosition</code></td><td>Starting character position of the character range to highlight.</td></tr><tr><td><code>endPosition</code></td><td>Ending character position of the character range to highlight.</td></tr></table>	<code>fld</code>	Pointer to a field object (FieldType data structure)	<code>startPosition</code>	Starting character position of the character range to highlight.	<code>endPosition</code>	Ending character position of the character range to highlight.
<code>fld</code>	Pointer to a field object (FieldType data structure)						
<code>startPosition</code>	Starting character position of the character range to highlight.						
<code>endPosition</code>	Ending character position of the character range to highlight.						
<b>Result</b>	Returns nothing.						
<b>Comments</b>	<p>To cancel a selection, set both <code>startPosition</code> and <code>endPosition</code> to the same value.</p> <p>If <code>startPosition</code> equals <code>endPosition</code>, then the current selection is unhighlighted.</p>						

## FldSetText

**Purpose** Set the text value of the field, update the word-wrapping information, and place the insertion point after the last visible character.

**Prototype**

```
void FldSetText (FieldPtr fld,  
                VoidHand textHandle,  
                Word offset,  
                Word size)
```

**Parameters**

fld	Pointer to a field object (FieldType data structure).
textHandle	Handle of a block containing a null-terminated text string.
offset	Offset from start of block to start of the text string.
size	Allocated size of text string, <i>not</i> the string length.

**Result** Returns nothing.

**Comments** The pointer passed is stored in the field's structure; in other words, this function doesn't make a copy of the string passed.

If a size of zero is passed, the size is computed as the block size, minus the offset passed. If more text is set than there is room for in memory, an error occurs.

---

**Warning:** This routine doesn't free the memory block that holds the current text value.

---

**See Also** [FldSetTextPtr](#), [FldSetTextHandle](#)



## **FldSetTextPtr**

**Purpose** Set the field's text to point to a text string.

**Prototype** `void FldSetTextPtr (FieldPtr fld, CharPtr textP)`

**Parameters** `fld` Pointer to a field object (FieldType data structure).  
`textP` Pointer to a null-terminated string.

**Result** Returns nothing.

**Comments** Since the field cannot resize a pointer (only handles can be resized), the field must be not editable and must be single line. If the field is editable or has more than one line, an error occurs.

This function does **not** visually update the field.

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

## **FldSetUsable**

**Purpose** Set a field usable or nonusable.

**Prototype** `void FldSetUsable (FieldPtr fld, Boolean usable)`

**Parameters** `fld` 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.

**See Also** [FldEraseField](#), [FldDrawField](#)

## **FldUndo**

**Purpose** Undo the last change made to the field object. Changes include typing, backspaces, delete, paste, and cut.

**Prototype** `void FldUndo (FieldPtr fld)`

**Parameters** `fld` Pointer to the field that has the focus.

**Result** Returns nothing.

**See Also** [FldPaste](#), [FldCut](#), [FldCopy](#)

## **FldWordWrap**

**Purpose** Given a string and a width, return the number of characters that can be displayed using the current font.

**Prototype** `Word FldWordWrap (CharPtr chars, Word maxWidth)`

**Parameters** `chars` Pointer to a null-terminated string.  
`maxWidth` Maximum line width in pixels.

**Result** Returns the number of characters.

## Font Functions

This section describes the functions used to manipulate fonts, and includes a summary of new font functions provided by version 3.0 of the Palm OS.

### New Font Features in Palm OS 3.0

The Palm OS 3.0 ROM provides a new font (`largeBoldFont`), two new font manipulation routines ([FontSelect](#) and [FntDefineFont](#)), and support for the use of custom fonts.

To use the large, bold font, pass the `largeBoldFont` selector to the [FntSetFont](#) function. Under Palm OS 3.0, if you try to draw with a font that isn't installed, the system uses the standard font by default. Note that previous versions of Palm OS can crash if told to use a nonexistent font.

The [FontSelect](#) function displays a dialog box in which the user can specify the use of one of the three primary fonts `stdFont`, `boldFont`, or `largeBoldFont`. For more information, see the description of [FontSelect](#) beginning on page 271.

The [FntDefineFont](#) function makes a custom font available to your application. For more information, see the description of [FntDefineFont](#) beginning on page 266.

Currently, Palm has not made available any tools or specifications to convert desktop fonts for use on Palm OS 3.0. If you have an urgent need for such support, send email to [devsupp@palm.com](mailto:devsupp@palm.com) for updated information.

#### **FntAccentHeight**

---

Not currently implemented.

---

#### **FntAscent**

---

Not currently implemented.

---

## **FntAverageCharWidth**

- Purpose** Return the average character width in the current font.
- Prototype** `short FntAverageCharWidth (void)`
- Parameters** None.
- Result** Returns the average character width (in pixels).

## **FntBaseLine**

- Purpose** Return the distance from the top of character cell to the baseline for the current font.
- Prototype** `short FntBaseLine (void)`
- Parameters** None.
- Result** Returns the baseline of the font (in pixels).

## **FntCharHeight**

- Purpose** Return the character height, in the current font including accents and descenders.
- Prototype** `short FntCharHeight (void)`
- Parameters** None
- Result** Height of the characters in the current font, expressed in pixels.

## **FntCharsInWidth**

**Purpose** Find the number of characters in a string that fit within a passed width. Spaces at the end of a string are ignored and removed. Characters after a carriage return are ignored, the string is considered truncated.

**Prototype** `void FntCharsInWidth (const Char * string,  
                          SWord *stringWidthP,  
                          SWord *stringLengthP,  
                          Boolean *fitWithinWidth)`

**Parameters**

<code>string</code>	Pointer to the char string.
<code>stringWidthP</code>	Maximum width to allow.
<code>stringLengthP</code>	Maximum characters to allow (assumes current Font).
<code>fitWithinWidth</code>	Set to TRUE if string is considered truncated.

**Result** When the call is completed, the information is updated as follows:

<code>stringWidthP</code>	Set to the width of the chars allowed.
<code>stringLengthP</code>	Set to the number of chars within the width.
<code>fitWithinWidth</code>	TRUE if the string is considered truncated, FALSE if it isn't.

### **FntCharsWidth**

**Purpose** Return the width of the specified character string. The Missing Character Symbol is substituted for any character which does not exist in the current font.

**Prototype** `SWord FntCharsWidth (const Char * chars, Word len)`

**Parameters**

<code>chars</code>	Pointer to a string of characters.
<code>len</code>	Number of character in the string.

**Result** Returns the width of the string, in pixels.

### **FntCharWidth**

**Purpose** Return the width of the specified character. If the specified character does not exist within the current font, the Missing Character Symbol is substituted.

**Prototype** `SWord FntCharWidth (const Char ch)`

**Parameters**

<code>ch</code>	Character whose width is needed.
-----------------	----------------------------------

**Result** Returns the width of the specified character (in pixels).

## **FntDefineFont**

**Purpose** Makes a custom font available to your application. 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.

**Prototype** `Err FntDefineFont (FontID font, FontPtr fontP)`

**Parameters**

<code>font</code>	An application-defined value greater than 128 that identifies the custom font to the system. Although this value is local to the application that called the <code>FntDefineFont</code> function, it must be greater than 128 because values less than 128 are reserved for system use.
<code>fontP</code>	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**

<code>0</code>	no error
<code>memErrNotEnoughSpace</code>	Insufficient dynamic heap space

**Comments** 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. This mechanism and its associated tools may be augmented in the near future; for more information, stay in contact with Palm.

**See Also** [FontSelect](#), [FntSetFont](#)

### **FntDescenderHeight**

**Purpose** Return the height of a character's descender in the current font. The height of a descender is the distance between the base line and the bottom of the character cell.

**Prototype** `SWord FntDescenderHeight (void)`

**Parameters** None.

**Result** Returns the height of a descender, expressed in pixels.

### **FntGetFont**

**Purpose** Return the Font ID of the current font.

**Prototype** `FontID FntGetFont (void)`

**Parameters** None.

**Result** Returns the Font ID of the current font.

### **FntGetFontPtr**

**Purpose** Return a pointer to the current font.

**Prototype** `FontPtr FntGetFontPtr (void)`

**Parameters** None.

**Result** Returns the `FontPtr` of the current font.

## **FntGetScrollValues**

**Purpose** Return the values needed to update a scroll bar based on a specified string and the position within the string.

**Prototype** `void FntGetScrollValues (const Char * const chars,  
const Word width,  
const Word scrollPos,  
const WordPtr linesP,  
const WordPtr topLine)`

**Parameters**

<code>chars</code>	Null-terminated string.
<code>width</code>	Width to word wrap at, in pixels.
<code>scrollPos</code>	Character position of the first visible character.
<code>linesP</code>	(returned) number of lines of text.
<code>topLine</code>	(returned) top visible line.

**Result** Returns nothing. Stores the number of lines of text in `linesP` and the top visible line in `topLine`.

## **FntLineHeight**

**Purpose** Return 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).

**Prototype** `short FntLineHeight (void)`

**Parameters** None.

**Result** Returns the height of a line in the current font.

## **FntLineWidth**

**Purpose** Return the width of the specified line of text, taking tab characters in to 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.

**Prototype** `SWord FntLineWidth (const Char * pChars,  
Word length)`

**Parameters**

<code>pChars</code>	Pointer to a string of characters.
<code>length</code>	Number of character in the string.

**Result** Returns the line width (in pixels).

## **FntSetFont**

**Purpose** Set the current font.

**Prototype** `FontID FntSetFont (FontID fontID)`

**Parameters**

<code>fontID</code>	ID of the font to make the active font.
---------------------	---

**Result** Returns the ID of the current font before the change.

### **FntWordWrap**

**Purpose** Given a string, determine the number of characters that can be displayed within the specified width.

**Prototype** `Word FntWordWrap (const Char * const chars,  
const Word maxWidth)`

**Parameters**

<code>chars</code>	Pointer to a null-terminated string.
<code>maxWidth</code>	Maximum line width in pixels.

**Result** Returns the length of the line, in characters.

### **FntWordWrapReverseNLines**

**Purpose** Word wrap 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.

**Prototype** `void FntWordWrapReverseNLines (  
const Char * const chars,  
const Word maxWidth,  
const WordPtr linesToScrollP,  
const WordPtr scrollPosP)`

**Parameters**

<code>chars</code>	Pointer to a null-terminated string.
<code>maxWidth</code>	Maximum line width in pixels.
<code>linesToScrollP</code>	Passed: lines to scroll returned: lines scrolled.
<code>scrollPosP</code>	Passed: first character returned: first character after wrapping.

**Result** Returns nothing. Stores the first character after wrapping and the number of lines scrolled in `scrollPosP` and `linesToScrollP`.

## **FontSelect**

**Purpose** Displays a dialog box in which the user can choose one of three system-supplied fonts, and returns a `FontID` value representing the user's choice.

**Prototype** `FontID FontSelect (FontID fontID)`

**Parameters** `fontID` A `fontID` value specifying the font to be highlighted as the default choice in the dialog box this function displays. This value must be one of the following system-supplied constants:

<code>stdFont</code>	Standard plain text font
<code>boldFont</code>	Bold version of <code>stdFont</code>
<code>largeBoldFont</code>	Larger version of <code>boldFont</code>

**Result** Returns a `fontID` value representing the font that the user chose in the dialog box this function displays.

**See Also** [FntGetFont](#), [FntSetFont](#)

## Form Functions

### FrmAlert

- Purpose** Create a modal dialog from an alert resource and display it until the user selects a button in the dialog.
- Prototype** `Word FrmAlert (Word 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).
- See Also** [FrmDoDialog](#), [FrmCustomAlert](#)

### FrmCloseAllForms

- Purpose** Send a [frmCloseEvent](#) to all open forms.
- Prototype** `void FrmCloseAllForms (void)`
- Parameters** None.
- 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.

**Prototype**

```
void FrmCopyLabel (FormPtr frm,  
                  Word labelID,  
                  CharPtr newLabel)
```

**Parameters**

<code>frm</code>	Pointer to memory block that contains the form.
<code>labelID</code>	ID of form label object.
<code>newLabel</code>	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. Redraw the label if the form's `usable` attribute and the label's `visible` attribute are set.

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

## **FrmCopyTitle**

**Purpose** Copy the title passed over the form's current title. If the form is visible, the new title is drawn.

**Prototype** `void FrmCopyTitle (FormPtr frm, CharPtr newTitle)`

**Parameters**

<code>frm</code>	Memory block that contains the form.
<code>newTitle</code>	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 string to be assigned dynamically.

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

## FrmCustomAlert

- Purpose** Create a modal dialog from an alert resource and display the dialog until the user taps a button in the alert dialog.
- Prototype** `Word FrmCustomAlert (Word alertId, CharPtr s1, CharPtr s2, CharPtr s3)`
- Parameters**
- |                         |  |
|-------------------------|--|
| <code>alertId</code>    | Resource ID of the alert.                        |
| <code>s1, s2, s3</code> | Strings to replace ^1, ^2, and ^3 (see Comment). |
- Result** Returns the button number the user tapped (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.
- See Also** [FrmAlert](#), [FrmDoDialog](#)

## **FrmDeleteForm**

**Purpose** Release the memory occupied by a form. Any memory allocated to objects in the form is also released.

**Prototype** `void FrmDeleteForm (FormPtr frm)`

**Parameters** `frm` Pointer to memory block that contains the form.

**Result** Returns nothing.

**Comments** This function doesn't modify the display.

**See Also** [FrmInitForm](#), [FrmReturnToForm](#)

## **FrmDispatchEvent**

**Purpose** Dispatch an event to the application's handler for the form.

**Prototype** `Boolean FrmDispatchEvent (EventPtr eventP)`

**Parameters** `eventP` Pointer to an event.

**Result** Returns nothing.

**Comments** The event is dispatched to the current form unless the form ID is specified in the event data, as, for example, with [frmOpenEvent](#).

**See Also** [FrmSetEventHandler](#), [FrmHandleEvent](#)

## FrmDoDialog

- Purpose** Display a modal dialog until the user taps a button in the dialog.
- Prototype** `Word FrmDoDialog (FormPtr frm)`
- Parameters** `frm` Pointer to memory block that contains the form.
- Result** Returns the number of the button the user tapped (first button is zero).
- Comments** A button's item number is determined by its order in the alert template; the first button has an item number of 0 (zero).
- See Also** [FrmInitForm](#), [FrmCustomAlert](#)

## FrmDrawForm

- Purpose** Draw all objects in a form and the frame around the form.
- Prototype** `void FrmDrawForm (FormPtr frm)`
- Parameters** `frm` Pointer to the memory block that contains the form.
- Result** Returns nothing.
- Comments** Saves the bits behind the form using the `bitsBehindForm` field.
- See Also** [FrmEraseForm](#), [FrmInitForm](#)

## **FrmEraseForm**

**Purpose** Erase a form from the display.

**Prototype** `void FrmEraseForm (FormPtr frm)`

**Parameters** `frm` Pointer to the memory block that contains the form.

**Result** Returns nothing.

**Comments** If the region obscured by the form was saved by [FrmDrawForm](#), this function restores that region.

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

## **FrmGetActiveForm**

**Purpose** Return the currently active form.

**Prototype** `FormPtr FrmGetActiveForm (void)`

**Parameters** None.

**Result** Returns the pointer to the memory block that contains the form.

**See Also** [FrmGetActiveFormID](#), [FrmSetActiveForm](#)

## FrmGetActiveFormID

- Purpose** Return the ID of the currently active form.
- Prototype** `Word FrmGetActiveFormID (void)`
- Parameters** None.
- Result** Returns the currently active form's ID number.
- See Also** [FrmGetActiveForm](#)

## FrmGetControlGroupSelection

- Purpose** Return the item number of the control selected in a group of controls.
- Prototype** `Byte FrmGetControlGroupSelection ( FormPtr frm,  
Byte groupNum)`
- Parameters**
- |                       |   |
|-----------------------|---|
| <code>frm</code>      | Pointer to memory block that contains the form. |
| <code>groupNum</code> | Control group number.                           |
- Result** Returns the item number of the selected control; returns 255 if no item is selected.
- Comments** The item number is the index into the form object's data structure.
- See Also** [FrmGetObjectId](#), [FrmGetObjectPtr](#), [FrmSetControlGroupSelection](#)

## **FrmGetControlValue**

**Purpose** Return the on/off state of a control.

**Prototype** `short FrmGetControlValue (FormPtr frm,  
Word objIndex)`

**Parameters**

<code>frm</code>	Pointer to memory block that contains the form.
<code>objIndex</code>	Item number of the object.

**Result** Returns the state of the control: 1 = on; 0 = off.

**Comments** The caller must specify a valid index. This function is used only for push button and check box control objects.

**See Also** [FrmGetObjectIndex](#), [FrmSetControlValue](#)

## **FrmGetFirstForm**

**Purpose** Return the first form in the window list.

**Prototype** `FormPtr FrmGetFirstForm (void)`

**Parameters** None.

**Result** Returns a pointer to a form, 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 (UI element) that has the focus.

**Prototype** `Word FrmGetFocus (FormPtr frm)`

**Parameters** `frm` Pointer to memory block that contains the form.

**Result** Returns the index of the object (UI element) that has the focus, or -1 if none does.

**See Also** [FrmGetObjectId](#), [FrmGetObjectPtr](#), [FrmSetFocus](#)

## FrmGetFormBounds

**Purpose** Return the visual bounds of the form; the region returned includes the form's frame.

**Prototype** `void FrmGetFormBounds (FormPtr frm, RectanglePtr r)`

**Parameters** `frm` Pointer to memory block that contains the form.

`r` Pointer to a `RectangleType` structure that will contain the bounds.

**Result** Returns the bounds of the form in `r`.

#### **FrmGetFormId**

**Purpose** Return the resource ID of a form.

**Prototype** `Word FrmGetFormId (FormPtr frm)`

**Parameters** `frm` Pointer to memory block that contains the form.

**Result** Returns form resource ID.

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

#### **FrmGetFormPtr**

**Purpose** Return a pointer to the form that has the specified ID.

**Prototype** `FormPtr FrmGetFormPtr (Word formId)`

**Parameters** `formId` Form ID number.

**Result** Returns a pointer to the memory block that contains the form, 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.
- Prototype** `VoidPtr FrmGetGadgetData (FormPtr frm,  
Word objIndex)`
- Parameters**
- |                       |   |
|-----------------------|---|
| <code>frm</code>      | Pointer to memory block that contains the form. |
| <code>objIndex</code> | Item number of the gadget object.               |
- 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](#)

## FrmGetLabel

- Purpose** Return pointer to the text of the specified label object in the specified form.
- Prototype** `CharPtr FrmGetLabel (FormPtr frm, Word labelID)`
- Parameters**
- |                      |   |
|----------------------|---|
| <code>frm</code>     | Pointer to memory block that contains the form. |
| <code>labelID</code> | 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.

**Prototype** `Word FrmGetNumberOfObjects (FormPtr frm)`

**Parameters** `frm` Pointer to memory block that contains the form.

**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.

**Prototype** `void FrmGetObjectBounds (FormPtr frm,  
Word ObjIndex,  
RectanglePtr r)`

**Parameters** `frm` Pointer to memory block that contains the form.  
`ObjIndex` Index of an object in the form.  
`r` Pointer to the rectangle containing the object bounds.

**Result** Returns nothing. The object's bounds are returned in `r`.

**See Also** [FrmGetObjectPosition](#), [FrmGetObjectIndex](#),  
[FrmSetObjectPosition](#)

## FrmGetObjectId

- Purpose** Return the ID of the specified object.
- Prototype** `Word FrmGetObjectId (FormPtr frm, Word objIndex)`
- Parameters**
- |                       |   |
|-----------------------|---|
| <code>frm</code>      | Pointer to memory block that contains the form. |
| <code>objIndex</code> | Index of an object in the form.                 |
- Result** Returns the ID number of a object.
- Comments** The application developer specifies a unique object ID.
- See Also** [FrmGetObjectPtr](#), [FrmGetObjectIndex](#)

## FrmGetObjectIndex

- Purpose** Return the item number of an object. The item number is the position of the object in the form's objects list.
- Prototype** `Word FrmGetObjectIndex (FormPtr frm, Word objID)`
- Parameters**
- |                    |   |
|--------------------|---|
| <code>frm</code>   | Pointer to memory block that contains the form. |
| <code>objID</code> | ID of an object in the form.                    |
- Result** Returns the item number of an object (the first item number is 0).
- See Also** [FrmGetObjectPtr](#), [FrmGetObjectId](#)

## **FrmGetObjectPosition**

**Purpose** Return the coordinate of the specified object relative to the form.

**Prototype**

```
void FrmGetObjectPosition (FormPtr frm,  
                           Word objIndex,  
                           SWordPtr x,  
                           SWordPtr y)
```

**Parameters**

frm	Pointer to memory block that contains the form.
objIndex	Item number of the object.
x, y	Pointer to window-relative x and y position.

**Result** Returns nothing.

**See Also** [FrmGetObjectBounds](#), [FrmSetObjectPosition](#)

## **FrmGetObjectPtr**

**Purpose** Return a pointer to the data structure of an object in a form.

**Prototype**

```
void * FrmGetObjectPtr (FormPtr frm,  
                        Word objIndex)
```

**Parameters**

frm	Pointer to memory block that contains the form.
objIndex	Item number of the object.

**Result** Returns a pointer to an object in the form.

**See Also** [FrmGetObjectIndex](#), [FrmGetObjectId](#)

## FrmGetObjectType

- Purpose** Return the type of an object.
- Prototype** `FormObjectKind FrmGetObjectType (FormPtr frm,  
Word objIndex)`
- Parameters**
- |                       |   |
|-----------------------|---|
| <code>frm</code>      | Pointer to memory block that contains the form. |
| <code>objIndex</code> | Item number of the object.                      |
- Result** Returns `FormObjectKind` of the item specified.

## FrmGetTitle

- Purpose** Return a pointer to the title string of a form.
- Prototype** `CharPtr FrmGetTitle (FormPtr frm)`
- Parameters**
- |                  |   |
|------------------|---|
| <code>frm</code> | Pointer to memory block that contains the form. |
|------------------|---|
- Result** Returns a pointer to title string.
- Comments** This is a pointer to the internal structure itself, *not* to a copy.
- See Also** [FrmCopyTitle](#), [FrmSetTitle](#)

## **FrmGetUserModifiedState**

**Purpose** Return TRUE if an object in the form has been modified by the user since it was initialized or since the last call to [FrmSetNotUserModified](#).

**Prototype** Boolean FrmGetUserModifiedState (FormPtr frm)

**Parameters** frm                      Pointer to the memory block that contains the form.

**Result** Returns TRUE if an object was modified; FALSE otherwise.

**Comments** Returns TRUE if the dirty attribute of the form has been set.

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

## **FrmGetWindowHandle**

**Purpose** Return the window handle of a form.

**Prototype** WinHandle FrmGetWindowHandle (FormPtr frm)

**Parameters** frm                      Pointer to memory block that contains the form.

**Result** Returns the handle of the memory block that the form is in. Since the form structure begins with the WindowType structure, this is also a WinHandle.

## FrmGotoForm

- Purpose** Send a [frmCloseEvent](#) to the current form; send a [frmLoadEvent](#) and a [frmOpenEvent](#) to the specified form.
- Prototype** `void FrmGotoForm (Word formId)`
- Parameters** `formId` ID of the form to display.
- Result** Returns nothing.
- Comments** The 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.
- Prototype** `Boolean FrmHandleEvent (FormPtr frm,  
EventPtr event)`
- Parameters** `frm` Pointer to the memory block that contains the form.  
`event` Pointer to the event data structure.
- Result** Returns TRUE if the event was handled.
- See Also** [FrmDispatchEvent](#)

#### **FrmHelp**

**Purpose** Display the specified help message until the user taps the Done button in the help dialog.

**Prototype** `void FrmHelp (Word helpMsgId)`

**Parameters** `helpMsgId` Resource ID of help message string.

**Result** Returns nothing.

**Comments** The ID passed is the resource ID of a string resource that contains the help message. The help message is displayed in a modal dialog that has vertical scrolls if necessary.

#### **FrmHideObject**

**Purpose** Erase the specified object and set its attribute data so that it does not redraw or respond to the pen.

**Prototype** `void FrmHideObject (FormPtr frm, Word objIndex)`

**Parameters** `frm` Pointer to memory block that contains the form.  
`objIndex` Item number of the object.

**Result** Returns nothing.

**See Also** [FrmGetObjectIndex](#), [FrmShowObject](#)

## **FrmInitForm**

- Purpose** Load and initialize a form resource.
- Prototype** `FormPtr FrmInitForm (Word rscID)`
- Parameters** `rscID` Resource ID of the form.
- Result** Returns a pointer to the form memory block.  
Displays an error message if the form has already been initialized.
- Comments** This function does not affect the display nor make the form active.
- See Also** [FrmDoDialog](#), [FrmDeleteForm](#)

## FrmNewBitmap

**Purpose** Create a new form bitmap dynamically.

**Prototype** `FormBitmapType * FrmNewBitmap (VoidPtr *formPP,  
const Word ID,  
const Word rscID,  
const Word x,  
const Word y)`

**Parameters**

<code>&lt;--&gt; formPP</code>	Pointer to the pointer to the form in which the new bitmap is installed. This value is not a handle; that is, the old <code>formPP</code> value is not necessarily valid after this function returns. In subsequent calls, always use the new <code>formPP</code> value returned by this function.
<code>ID</code>	Symbolic ID of the bitmap, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
<code>rscID</code>	Numeric value identifying the resource that provides the bitmap. This value must be unique within the application scope.
<code>x</code>	Horizontal coordinate of the upper-left corner of the bitmap's boundaries, relative to the window in which it appears.
<code>y</code>	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.

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

## **FrmNewForm**

**Purpose** Create a new form object dynamically.

**Prototype**

```
FormPtr FrmNewForm (const Word formID,  
                    const Char * const titleStrP,  
                    Word x,  
                    Word y,  
                    Word width,  
                    Word height,  
                    const Boolean modal,  
                    const Word defaultButton,  
                    const Word helpRscID,  
                    const Word menuRscID)
```

<b>Parameters</b>	<code>formID</code>	Symbolic ID of the form, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
	<code>titleStrP</code>	Pointer to a string that is the title of the form.
	<code>x</code>	Horizontal coordinate of the upper-left corner of the form's boundaries, relative to the window in which it appears.
	<code>y</code>	Vertical coordinate of the upper-left corner of the form's boundaries, relative to the window in which it appears.
	<code>width</code>	Width of the form, expressed in pixels. Valid values are 1 - 160.
	<code>height</code>	Height of the form, expressed in pixels. Valid values are 1 - 160.
	<code>modal</code>	TRUE specifies that the form ignores pen events outside its boundaries.
	<code>defaultButton</code>	Symbolic ID of the button that provides the form's default action, specified by the developer.
	<code>helpRscID</code>	Symbolic ID of the resource that provides the form's online help, specified by the developer.

## Palm OS User Interface Functions

### Form Functions

---

under Palm OS 3.0, only modal dialogs 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.

**See Also** [FrmValidatePtr](#), [WinValidateHandle](#), [FrmRemoveObject](#)

## FrmNewGadget

<b>Purpose</b>	Create a new gadget dynamically and install it in the specified form.												
<b>Prototype</b>	<pre>FormGadgetType * FrmNewGadget (VoidPtr *formPP,                                 const Word id,                                 const Word x,                                 const Word y,                                 const Word width,                                 const Word height)</pre>												
<b>Parameters</b>	<table><tr><td><code>&lt;--&gt; formPP</code></td><td>Pointer to the pointer to the form in which the new gadget is installed. This value is not a handle; that is, the old <code>formPP</code> value is not necessarily valid after this function returns. In subsequent calls, always use the new <code>formPP</code> value returned by this function.</td></tr><tr><td><code>id</code></td><td>Symbolic ID of the gadget, specified by the developer. By convention, this ID should match the resource ID (not mandatory).</td></tr><tr><td><code>x</code></td><td>Horizontal coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.</td></tr><tr><td><code>y</code></td><td>Vertical coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.</td></tr><tr><td><code>width</code></td><td>Width of the gadget, expressed in pixels. Valid values are 1 - 160.</td></tr><tr><td><code>height</code></td><td>Height of the gadget, expressed in pixels. Valid values are 1 - 160.</td></tr></table>	<code>&lt;--&gt; formPP</code>	Pointer to the pointer to the form in which the new gadget is installed. This value is not a handle; that is, the old <code>formPP</code> value is not necessarily valid after this function returns. In subsequent calls, always use the new <code>formPP</code> value returned by this function.	<code>id</code>	Symbolic ID of the gadget, specified by the developer. By convention, this ID should match the resource ID (not mandatory).	<code>x</code>	Horizontal coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.	<code>y</code>	Vertical coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.	<code>width</code>	Width of the gadget, expressed in pixels. Valid values are 1 - 160.	<code>height</code>	Height of the gadget, expressed in pixels. Valid values are 1 - 160.
<code>&lt;--&gt; formPP</code>	Pointer to the pointer to the form in which the new gadget is installed. This value is not a handle; that is, the old <code>formPP</code> value is not necessarily valid after this function returns. In subsequent calls, always use the new <code>formPP</code> value returned by this function.												
<code>id</code>	Symbolic ID of the gadget, specified by the developer. By convention, this ID should match the resource ID (not mandatory).												
<code>x</code>	Horizontal coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.												
<code>y</code>	Vertical coordinate of the upper-left corner of the gadget's boundaries, relative to the window in which it appears.												
<code>width</code>	Width of the gadget, expressed in pixels. Valid values are 1 - 160.												
<code>height</code>	Height of the gadget, expressed in pixels. Valid values are 1 - 160.												
<b>Result</b>	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.												
<b>Comments</b>	A gadget is a custom UI object. For more information, see "Gadget Resource" starting on page 110.												
<b>See Also</b>	<a href="#">FrmRemoveObject</a>												

### FrmNewLabel

**Purpose** Create a new label object dynamically and install it in the specified form.

**Prototype**

```
FormLabelType* FrmNewLabel (  
    VoidPtr *formPP,  
    const Word ID,  
    const Char * const textP,  
    const Word x,  
    const Word y,  
    const FontID font)
```

<b>Parameters</b>	<--> formPP	Pointer to the 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. In subsequent calls, always use the new formPP value returned by this function.
	ID	Symbolic ID of the label, specified by the developer. By convention, this ID should match the resource ID (not mandatory).
	textP	Pointer to a string that provides the label text.
	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.

**See Also** [CtlValidatePointer](#), [FrmRemoveObject](#)

## FrmPointInTitle

**Purpose** Returns TRUE if the coordinate passed is within the bounds of form's title.

**Prototype** Boolean FrmPointInTitle (FormPtr frm,  
Short x,  
Short y)

**Parameters** frm Memory block that contains the form.  
x, y Window-relative x and y coordinate.

**Result** Returns TRUE if the specified coordinate is in the form's title.

## FrmPopupForm

**Purpose** Send a [frmOpenEvent](#) to the specified form. This routine differs from [FrmGotoForm](#) in that the current form is not closed.

**Prototype** void FrmPopupForm (Word formId)

**Parameters** formID Resource ID of form to open.

**Result** Returns nothing.

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

## FrmRemoveObject

**Purpose** Remove the specified object from the specified form.

**Prototype** `Err FrmRemoveObject (FormPtr *formPP,  
Word objIndex)`

**Parameters**

<code>&lt;--&gt; formPP</code>	Pointer to the pointer to the form from which this function removes an object. This value is not a handle; that is, the old <code>formPP</code> value is not necessarily valid after this function returns. In subsequent calls, always use the new <code>formPP</code> value returned by this function.
<code>objIndex</code>	The object to remove, specified as an index into the list of objects installed in the form. You can use the <a href="#">FrmGetObjectIndex</a> function to discover this value.

**Result** Returns 0 if no error.

**Comments** You can use this function to remove any form object (bitmaps, controls, lists, and so on) and free the memory allocated to it.

**See Also** [FrmNewBitmap](#), [FrmNewForm](#), [FrmNewGadget](#), [FrmNewLabel](#), [CtlNewControl](#), [FldNewField](#), [LstNewList](#)

## FrmReturnToForm

- Purpose** Erase and delete the currently active form and make the specified form the active form.
- Prototype** `void FrmReturnToForm (Word 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.
- See Also** [FrmGotoForm](#), [FrmPopupForm](#)

## FrmSaveAllForms

- Purpose** Send a [frmSaveEvent](#) to all open forms.
- Prototype** `void FrmSaveAllForms (void)`
- Parameters** None.
- Result** Returns nothing.
- See Also** [FrmCloseAllForms](#)

## **FrmSetActiveForm**

**Purpose** Set the active form. All input (key and pen) is directed to the active form.

**Prototype** `void FrmSetActiveForm (FormPtr frm)`

**Parameters** `frm` Pointer to memory block that contains the form.

**Result** Returns nothing.

**Comments** A [penDownEvent](#) outside the form but within the display area is ignored.

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

## **FrmSetCategoryLabel**

**Purpose** Set the category label displayed on the title line of a form. If the form's `visible` attribute is set, redraw the label.

**Prototype** `void FrmSetCategoryLabel (FormPtr frm,  
Word objIndex,  
CharPtr newLabel)`

**Parameters** `frm` Pointer to memory block that contains the form.

`objIndex` Item number of the object.

`newLabel` Pointer to the name of the new category.

**Result** Returns nothing.

**Comments** The pointer to the new label is saved in the object.

## FrmSetControlGroupSelection

<b>Purpose</b>	Set the selected control in a group of controls.	
<b>Prototype</b>	void FrmSetControlGroupSelection (FormPtr frm, Byte groupNum, Word controlId)	
<b>Parameters</b>	frm	Pointer to memory block that contains the form.
	groupNum	Control group number.
	controlID	ID of control to set.
<b>Result</b>	Returns nothing.	
<b>Comments</b>	Function unsets all the other controls in the group. The display is updated.	
<b>See Also</b>	<a href="#">FrmGetControlGroupSelection</a>	

## FrmSetControlValue

<b>Purpose</b>	Turn a control on or off.	
<b>Prototype</b>	void FrmSetControlValue (FormPtr frm, Word objIndex, short newValue)	
<b>Parameters</b>	frm	Pointer to memory block that contains the form.
	objIndex	Item number of the object.
	newValue	New control value (non-zero equals on).
<b>Result</b>	Returns nothing.	
<b>Comments</b>	The display is not changed.	
<b>See Also</b>	<a href="#">FrmGetControlValue</a>	

## FrmSetEventHandler

**Purpose** Set the event handler callback routine for the specified form.

**Prototype** `void FrmSetEventHandler (FormPtr frm,  
FormEventHandlerPtr handler)`

**Parameters**

<code>frm</code>	Pointer to memory block that contains the form.
<code>handler</code>	Address of a function.

**Result** Returns nothing.

**Comments** [FrmHandleEvent](#) calls this handler whenever it receives an event. This routine should be called right after a form resource is loaded. The callback routine is the mechanism for dispatching events to an application. The tutorial explains how to use callback routines.

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

## FrmSetFocus

**Purpose** Set the focus of a form to the specified object.

**Prototype** `void FrmSetFocus (FormPtr frm, Word objIndex)`

**Parameters**

<code>frm</code>	Pointer to memory block that contains the form.
<code>objIndex</code>	Item number of the object (UI element) that gets the focus.

**Result** Returns nothing.

**See Also** [FrmGetFocus](#), [FrmGetObjectIndex](#)

## FrmSetGadgetData

- Purpose** Store the value passed in the data field of the gadget object.
- Prototype**

```
void FrmSetGadgetData (FormPtr frm,
                        Word objIndex,
                        VoidPtr data)
```
- Parameters**
- |                       |   |
|-----------------------|---|
| <code>frm</code>      | Pointer to memory block that contains the form. |
| <code>objIndex</code> | Item number of the object.                      |
| <code>data</code>     | Application-defined value.                      |
- Result** Returns nothing.
- 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** [FrmGetGadgetData](#), [FrmGetObjectIndex](#)

## FrmSetMenu

- Purpose** Change a form's menu bar.
- Prototype**

```
void FrmSetMenu (FormPtr frm, Word menuRscID)
```
- Parameters**
- |                        |                                      |
|------------------------|--------------------------------------|
| <code>frm</code>       | Memory block that contains the form. |
| <code>menuRscID</code> | Resource ID of the menu.             |
- Result** Returns nothing.

#### FrmSetNotUserModified

**Purpose** Clear the flag that keeps track of whether or not the form has been modified by the user.

**Prototype** `void FrmSetNotUserModified (FormPtr frm)`

**Parameters** `frm` Pointer to memory block that contains the form.

**Result** Returns nothing.

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

#### FrmSetObjectBounds

**Purpose** Set the bounds of the specified form object.

**Prototype** `void FrmSetObjectBounds (FormPtr frm,  
Word objIndex,  
RectanglePtr bounds)`

**Parameters** `frmPtr` Memory block that contains the form.  
`objIndex` Item number of the object.  
`bounds` Window-relative bounds.

**Result** Returns nothing.

## FrmSetObjectPosition

**Purpose** Set the window-relative coordinate of the specified object.

**Prototype**

```
void FrmSetObjectPosition (FormPtr frm,
                           Word objIndex,
                           SWord x,
                           SWord y)
```

**Parameters**

<code>frm</code>	Pointer to memory block that contains the form.
<code>objIndex</code>	Item number of the object.
<code>x</code>	Window-relative coordinate.
<code>y</code>	Window-relative coordinate.

**Result** Returns nothing.

**Comments** Doesn't update the display. Presently, only label objects are affected.

**Caveat** This function currently doesn't work when used on a bitmap.

**See Also** [FrmGetObjectPosition](#), [FrmGetObjectIndex](#),  
[FrmGetObjectBounds](#)

#### FrmSetTitle

**Purpose** Set the title of a form. If the form is visible, draw the new title.

**Prototype** `void FrmSetTitle (FormPtr frm, CharPtr newTitle)`

**Parameters**

<code>frm</code>	Pointer to memory block that contains the form.
<code>newTitle</code>	Pointer to the new title string.

**Result** Returns nothing.

**Comments** Draws the title if the form is visible.  
Saves the pointer to the passed title string. Does *not* make a copy.

**See Also** [FrmGetTitle](#), [FrmCopyTitle](#), [FrmCopyLabel](#)

#### FrmShowObject

**Purpose** Set an object (UI element) inside a form as usable. If the form is visible, draw the object.

**Prototype** `void FrmShowObject (FormPtr frm, Word objIndex)`

**Parameters**

<code>frm</code>	Pointer to memory block that contains the form.
<code>objIndex</code>	Item number of the object.

**Result** Returns nothing.

**See Also** [FrmHideObject](#), [FrmGetObjectIndex](#)

## FrmUpdateForm

**Purpose** Send a [frmUpdateEvent](#) to the specified form.

**Prototype** `void FrmUpdateForm (Word formId, Word updateCode)`

**Parameters**

<code>formId</code>	Resource ID of form to open.
<code>updateCode</code>	If the update code is <code>frmRedrawUpdateCode</code> , the form reinitializes its global variables and redraws itself. Otherwise, the form reinitializes its global variables but does not redraw itself.

**Result** Returns nothing.

## FrmUpdateScrollers

**Purpose** Visually update the field scroll arrow buttons.

**Prototype** `void FrmUpdateScrollers (FormPtr frm,  
Word upIndex,  
Word downIndex,  
Boolean scrollableUp,  
Boolean scrollableDown)`

**Parameters**

<code>frm</code>	Pointer to a form.
<code>upIndex</code>	Index of the up-scroller button.
<code>downIndex</code>	Index of the down-scroller button.
<code>scrollableUp</code>	TRUE if the up-scroll should be active.
<code>scrollableDown</code>	TRUE if the down-scroll should be active.

**Result** Returns nothing.

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

#### **FrmValidatePtr**

**Purpose** Return TRUE if the specified pointer references a valid form.

**Prototype** Boolean FrmValidatePtr (FormPtr frm)

**Parameters** frm                      Pointer to be tested.

**Result** Returns TRUE if the specified handle references a non-NULL pointer to an object having a valid form structure.

**Comments** This function is intended for debugging purposes only. Do not include it in commercial code.

#### **FrmVisible**

**Purpose** Return TRUE if the form is visible (is drawn).

**Prototype** Boolean FrmVisible (FormPtr frm)

**Parameters** frm                      Pointer to memory block that contains the form.

**Result** Returns TRUE if visible; FALSE if not visible.

**See Also** [FrmDrawForm](#), [FrmEraseForm](#)

## Character Attribute Functions

### GetCharAttr

**Purpose** Return a pointer to the character attribute. This array is used by the character classification and character conversion macros (such as `isalpha`).

**Prototype** `WordPtr GetCharAttr (void)`

**Parameters** None

**Result** A pointer to the attributes array. See `CharAttr.h` for an explanation of the attributes.

### GetCharCaselessValue

**Purpose** Return a pointer to an array that maps all characters to an assigned caseless and accentless value. Use this function for finding text.

**Prototype** `BytePtr GetCharCaselessValue (void)`

**Parameters** None.

**Result** Returns a pointer to the sort array.  
The compiler pads each byte out to a word so each index position contains two characters.

Note: `array[x].high = sort value for character 2x+1.`

## Palm OS User Interface Functions

### Character Attribute Functions

---

**Comment** The `GetCharCaselessValue` conversion table converts each character into a numeric value that is caseless and sorted according to Microsoft Windows sorting rules:

- Punctuation characters have the lowest values,
- followed by numbers,
- followed by alpha characters.

All forms of each alpha character have equivalent values, so that e = E = e-grave = e-circumflex, etc.

This conversion table is used by all the Palm OS sorting and comparison routines to yield caseless searches and caseless sorts in the same order as Windows-based programs.

### GetCharSortValue

**Purpose** Return a pointer to an array that maps all characters to an assigned sorting value. Use this function for ordering (sorting) text.

**Prototype** `BytePtr GetCharSortValue (void)`

**Parameters** None.

**Result** Returns a pointer to the attributes array.

The compiler pads each byte out to a word so each index position contains two characters.

Note: `array[x].low` = sort value for character `2x`.

## Graffiti Manager Functions

### GrfAddMacro

**Purpose** Add a macro to the macro list.

**Prototype** `Err GrfAddMacro ( CharPtr nameP,  
BytePtr macroDataP,  
Word dataLen)`

**Parameters**

<code>nameP</code>	Name of macro.
<code>macroDataP</code>	Data of macro.
<code>dataLen</code>	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](#)

### GrfAddPoint

**Purpose** Add a point to the Graffiti point buffer.

**Prototype** `Err GrfAddPoint (PointType* ptP)`

**Parameters**

<code>ptP</code>	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.

**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.

**Prototype** `Err GrfDeleteMacro (Word 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.
- Prototype** `Err GrfFilterPoints (void)`
- Parameters** None.
- Result** Always returns 0.
- See Also** [GrfMatch](#)

## GrfFindBranch

- Purpose** Locate a branch in the Graffiti dictionary by flags.
- Prototype** `Err GrfFindBranch (Word 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.

**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.

**Prototype** `Err GrfGetAndExpandMacro ( CharPtr nameP,  
BytePtr macroDataP,  
WordPtr dataLenP)`

**Parameters**

<code>nameP</code>	Name of macro to look up.
<code>macroDataP</code>	Macro contents returned here.
<code>dataLenP</code>	On entry, size of <code>macroDataP</code> 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.

**Prototype** `Err GrfGetGlyphMapping ( Word glyphID,  
WordPtr flagsP,  
void* dataPtrP,  
WordPtr dataLenP,  
WordPtr uncertainLenP)`

**Parameters**

<code>glyphID</code>	Glyph ID to look up.
<code>flagsP</code>	Returned dictionary flags.
<code>dataPtrP</code>	Where returned text goes.
<code>dataLenP</code>	On entry, size of <code>dataPtrP</code> ; on exit, number of bytes returned.
<code>uncertainLenP</code>	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.

**Prototype** `Err GrfGetMacro (CharPtr nameP,  
BytePtr macroDataP,  
WordPtr dataLenP)`

**Parameters**

<code>nameP</code>	Name of macro to lookup.
<code>macroDataP</code>	Macro contents returned here.
<code>dataLenP</code>	On entry: size of <code>macroDataP</code> 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.

**Prototype** `Err GrfGetMacroName (Word index, CharPtr nameP)`

**Parameters**

<code>index</code>	Index of macro.
<code>nameP</code>	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.

**Prototype** `Err GrfGetNumPoints (WordPtr 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.

**Prototype** `Err GrfGetPoint (Word 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.

**Prototype** `Err GrfGetState ( Boolean* capsLockP,  
Boolean* numLockP,  
WordPtr tempShiftP,  
Boolean* autoShiftedP)`

**Parameters**

<code>capsLockP</code>	Returns TRUE if caps lock on.
<code>numLockP</code>	Returns TRUE if num lock on.
<code>tempShiftP</code>	Current temporary shift.
<code>autoShiftedP</code>	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 Note** 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.
- 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.
- Prototype** `Err GrfMatch ( WordPtr flagsP,  
void* dataPtrP,  
WordPtr dataLenP,  
WordPtr uncertainLenP,  
GrfMatchInfoPtr matchInfoP)`
- Parameters**
- |                            |   |
|----------------------------|---|
| <code>flagsP</code>        | Glyph flags are returned here.  |
| <code>dataPtrP</code>      | Return text is placed here.   |
| <code>dataLenP</code>      | Size of <code>dataPtrP</code> on exit; number of characters returned on exit. |
| <code>uncertainLenP</code> | Return number of uncertain characters.  |
| <code>matchInfoP</code>    | Array of <code>grfMaxMatches</code> , or nil.                                 |
- Result** Returns 0 if no error, or `grfErrNoGlyphTable`, `grfErrNoDictionary`, or `grfErrNoMapping` if an error occurs.
- See Also** [GrfAddPoint](#), [GrfFlushPoints](#)

## **GrfMatchGlyph**

**Purpose** Recognize the current stroke as a glyph.

**Prototype** `Err GrfMatchGlyph ( GrfMatchInfoPtr matchInfoP,  
Word maxUncertainty,  
Word maxMatches )`

**Parameters**

<code>matchInfoP</code>	Pointer to array of matches to fill in.
<code>maxUncertainty</code>	Maximum number of errors to tolerate.
<code>maxMatches</code>	Size of <code>matchInfoP</code> 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.

**Prototype**

```
Err GrfProcessStroke (  PointType* startPtP,  
                        PointType* endPtP,  
                        Boolean upShift)
```

**Parameters**

<code>startPtP</code>	Start point of stroke.
<code>endPtP</code>	End point of stroke.
<code>upShift</code>	Set to <code>TRUE</code> 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` (documented in “Developing Palm OS Applications, Part II”)

## **GrfSetState**

**Purpose** Set the current shift state of Graffiti.

**Prototype** `Err GrfSetState (Boolean capsLock,  
Boolean numLock,  
Boolean upperShift)`

**Parameters**

<code>capsLock</code>	Set to TRUE to turn on caps lock.
<code>numLock</code>	Set to TRUE to turn on num lock.
<code>upperShift</code>	Set to TRUE to put into upper shift.

**Result** Always returns 0.

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

## **SysGrfShortCutListDialog**

**Purpose** Pop up the Graffiti ShortCut list as a field object with the focus.

**Prototype** `void SysGrfShortCutListDialog (void)`

**Parameters**

<code>event</code>	Pointer to an EventType structure.
--------------------	------------------------------------

**Result** The field's text chunk is changed.

**See Also** [GrfGetMacro](#), [GrfGetMacroName](#)



## **GsiEnabled**

**Purpose** Return `TRUE` if the Graffiti-shift state indicator is enabled, or `FALSE` if it's disabled.

**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.

**Prototype** `void GsiInitialize (void)`

**Parameters** None.

**Result** Returns nothing.

## **GsiSetLocation**

**Purpose** Set the display-relative position of the Graffiti-shift state indicator.

**Prototype** `void GsiSetLocation (short x, short 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**

<b>Purpose</b>	Set the Graffiti-shift state indicator.				
<b>Prototype</b>	<pre>void GsiSetShiftState ( Word lockFlags,                         Word tempShift)</pre>				
<b>Parameters</b>	<table><tr><td>lockFlags</td><td>glfCapsLock or glfNumLock.</td></tr><tr><td>tempShift</td><td>The current temporary shift.</td></tr></table>	lockFlags	glfCapsLock or glfNumLock.	tempShift	The current temporary shift.
lockFlags	glfCapsLock or glfNumLock.				
tempShift	The current temporary shift.				
<b>Result</b>	Returns nothing.				
<b>Comment</b>	This function affects only the state of the UI element, not the underlying Graffiti engine.				
<b>See Also</b>	<a href="#">GrfSetState</a>				

## 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.
- 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](#)

### InsPtEnabled

- Purpose** Return TRUE if the insertion point is enabled or FALSE if the insertion point is disabled.
- Prototype** `Boolean InsPtEnabled (void)`
- Parameters** None.
- Result** Returns TRUE if the insertion point is enabled (blinking); returns FALSE if the insertion point is disabled (invisible).
- See Also** [InsPtEnable](#)

## **InsPtGetHeight**

- Purpose** Return the height of the insertion point.
- Prototype** `short InsPtGetHeight (void)`
- Parameters** None.
- Result** Returns the height of the insertion point, in pixels.

## **InsPtGetLocation**

- Purpose** Return the screen-relative position of the insertion point.
- Prototype** `void InsPtGetLocation (short *x, short *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.
- Prototype** `void InsPtSetHeight (short 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](#)

## **InsPtSetLocation**

- Purpose** Set the screen-relative position of the insertion point.
- Prototype** `void InsPtSetLocation (short x, short 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](#)

## Functions for System Use Only

### InsPtCheckBlink

**Prototype** `void InsPtCheckBlink (void)`

---

WARNING: For System Use Only.

---

### InsPtInitialize

**Prototype** `void InsPtInitialize (void)`

---

WARNING: For System Use Only.

---

## Key Manager Functions

### KeyCurrentState

**Purpose** Return bit field with bits set for each key that is currently depressed.

**Prototype** `DWord KeyCurrentState (void)`

**Parameters** None.

**Result** Returns a DWord with bits set for keys that are depressed. See `keyBitPower`, `keyBitPageUp`, `keyBitPageDown`, etc., in `KeyMgr.h`.

**Comments** Called by applications that need to poll the keys.

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

## KeyRates

**Purpose** Get or set the key repeat rates.

**Prototype** `Err KeyRates ( Boolean set,  
WordPtr initDelayP,  
WordPtr periodP,  
WordPtr doubleTapDelayP,  
BooleanPtr queueAheadP)`

**Parameters**

<code>set</code>	If <code>TRUE</code> , settings are changed; if <code>FALSE</code> , current settings are returned.
<code>initDelayP</code>	Initial delay in ticks for a auto-repeat event.
<code>periodP</code>	Auto-repeat rate specified as period in ticks.
<code>doubleTapDelayP</code>	Maximum double-tap delay, in ticks.
<code>queueAheadP</code>	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](#)

## Functions for System Use Only

### KeyBootKeys

**Prototype** `DWord KeyBootKeys (void)`

---

WARNING: This function for use by system software only.

---



## List UI Functions

### LstDrawList

**Purpose** Draw the list object if it's usable. Set its `visible` attribute to `TRUE`.

**Prototype** `void LstDrawList (ListPtr list)`

**Parameters** `list` Pointer to list object (`ListType` data structure).

**Result** Returns nothing.

**Comments** If there are more choices than can be displayed, this function ensures that the current selection is visible. If possible, the current selection is displayed at the top. The current selection is highlighted. 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.

**Prototype** `void LstEraseList (ListPtr ListP)`

**Parameters** `ListP` Pointer to a list object (`ListType` data structure).

**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.

**Prototype** `Word LstGetNumberOfItems (ListPtr ListP)`

**Parameters** `ListP` Pointer to a list object (ListType data structure).

**Result** Returns the number of items in a list.

**See Also** [FrmGetObjectPtr](#), [LstSetListChoices](#)

## LstGetSelection

**Purpose** Return the currently selected choice in the list. If there is no selection, return `NoListSelection` (-1).

**Prototype** `Word LstGetSelection (ListPtr ListP)`

**Parameters** `ListP` Pointer to list object.

**Result** Returns the item number of the current list choice. The list choices are numbered sequentially, starting with 0; -1 = none.

**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.

**Prototype** `CharPtr LstGetSelectionText ( ListPtr ListP,  
Word itemNum)`

**Parameters** `ListP` Pointer to 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` structure, not a copy.

**See Also** [FrmGetObjectPtr](#), [LstSetListChoices](#)

## **LstGetVisibleItems**

**Purpose** Return the number of visible items.

**Prototype** `Int LstGetVisibleItems (ListPtr pList)`

**Parameters** `pList` Pointer to list object

**Result** The number of items visible.

## 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.
- Prototype** `Boolean LstHandleEvent ( ListPtr listP,  
EventPtr pEvent )`
- Parameters**
- |                     |   |
|---------------------|---|
| <code>listP</code>  | Pointer to a list object ( <code>ListType</code> data structure). |
| <code>pEvent</code> | Pointer to an <code>EventType</code> 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.

## **LstMakeItemVisible**

**Purpose** Make an item visible, preferably at the top. If the item is already visible, make no changes.

**Prototype** `LstMakeItemVisible ( ListPtr ListP,  
Word itemNum)`

**Parameters** `ListP` Pointer to a list object (`ListType` data structure).  
`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.

**Prototype** `Err LstNewList (VoidPtr *formPP,  
const Word id,  
const Word x,  
const Word y,  
const Word width,  
const Word height,  
const FontID font,  
const Word visibleItems,  
const Word 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.
	triggerId	Symbolic ID of the popup trigger associated with the new list. This ID is specified by the developer; by convention, this ID should match the resource ID (not mandatory).

**Result** Returns 0 if no error, or XXXXXXXX if an error occurs.

**See Also** [LstDrawList](#), [FrmRemoveObject](#)

## **LstPopupList**

**Purpose** Display a modal window that contains the items in the list.

**Prototype** `short LstPopupList (ListPtr ListP)`

**Parameters** `ListP` Pointer to 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.

**Prototype** `Boolean LstScrollList (ListPtr pList,  
enum directions direction,  
short itemCount)`

**Parameters** `pList` Pointer to 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.

## LstSetDrawFunction

- Purpose** Set a callback function to draw each item instead of drawing the item's text string.
- Prototype** `void LstSetDrawFunction (ListPtr list,  
ListDrawDataFuncPtr func)`
- Parameters**
- |                   |  |
|-------------------|--|
| <code>list</code> | Pointer to list object.                |
| <code>func</code> | Pointer to function which draws items. |
- Result** Returns nothing.
- Comments** This function also adjusts `topItem` to prevent a shrunken list from being scrolled down too far. Use this function for custom draw functionality.
- See Also** [FrmGetObjectPtr](#), [LstSetListChoices](#)

## LstSetHeight

- Purpose** Set the number of items visible in a list.
- Prototype** `void LstSetHeight ( ListPtr ListP,  
Word visibleItems)`
- Parameters**
- |                           |                                    |
|---------------------------|------------------------------------|
| <code>ListP</code>        | Pointer to list object.            |
| <code>visibleItems</code> | 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 strings passed to this function. This function doesn't affect the display of the list. If the list is visible, erases the old list items.

**Prototype**

```
void LstSetListChoices ( ListPtr ListP,  
                        char ** itemsText,  
                        UInt numItems)
```

**Parameters** `ListP` Pointer to a list object.  
`itemsText` Pointer to an array of text strings.  
`numItems` Number of choices in the list.

**Result** Returns nothing.

**See Also** [FrmGetObjectPtr](#), [LstSetSelection](#), [LstSetTopItem](#),  
[LstDrawList](#), [LstSetHeight](#), [LstSetDrawFunction](#)

## **LstSetPosition**

**Purpose** Set the position of a list.

**Prototype**

```
void LstSetPosition ( ListPtr ListP,  
                     short x,  
                     short y)
```

**Parameters** `ListP` Pointer to a list object  
`x, y` Left and top bound.

**Result** Returns nothing.

**Comments** List is not redrawn. Don't call this function when the list is visible.

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

## LstSetSelection

<b>Purpose</b>	Set the selection for a list.				
<b>Prototype</b>	<pre>void LstSetSelection ( ListPtr ListP,                       Word itemNum)</pre>				
<b>Parameters</b>	<table><tr><td>ListP</td><td>Pointer to a list object.</td></tr><tr><td>itemNum</td><td>Item to select (0 = first item in list; -1 = none).</td></tr></table>	ListP	Pointer to a list object.	itemNum	Item to select (0 = first item in list; -1 = none).
ListP	Pointer to a list object.				
itemNum	Item to select (0 = first item in list; -1 = none).				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	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.				
<b>See Also</b>	<a href="#">FrmGetObjectPtr</a>				

## LstSetTopItem

<b>Purpose</b>	Set the item visible. The item cannot become the top item if it's on the last page.				
<b>Prototype</b>	<pre>void LstSetTopItem (ListPtr ListP, UInt itemNum)</pre>				
<b>Parameters</b>	<table><tr><td>ListP</td><td>Pointer to list object.</td></tr><tr><td>itemNum</td><td>Item to select (0 = first item in list).</td></tr></table>	ListP	Pointer to list object.	itemNum	Item to select (0 = first item in list).
ListP	Pointer to list object.				
itemNum	Item to select (0 = first item in list).				
<b>Result</b>	Returns nothing.				
<b>Comments</b>	Does <i>not</i> update the display.				
<b>See Also</b>	<a href="#">FrmGetObjectPtr</a> , <a href="#">LstSetSelection</a> , <a href="#">LstMakeItemVisible</a> , <a href="#">LstDrawList</a> , <a href="#">LstEraseList</a>				

## Menu Functions

### MenuDispose

**Purpose** Release any memory allocated to support the menu management.

**Prototype** `void MenuDispose (MenuBarPtr MenuP)`

**Parameters** MenuP                      Pointer returned by [MenuInit](#); this is a pointer to a MenuBarType data structure.

**Result** Returns nothing.

**Comments** This function is useful for applications that have multiple menu bars. It frees all memory allocated by a menu, resets the command status, and restores the saved bits to the screen.

**See Also** [MenuInit](#), [MenuDrawMenu](#)

## MenuDrawMenu

- Purpose** Draw the current menu bar and the last pull-down that was visible.
- Prototype** `void MenuDrawMenu (MenuBarPtr MenuP)`
- Parameters** MenuP                      Pointer to a MenuBarType data structure.
- Result** Returns nothing.
- Comments** If a pull-down menu was visible the last time the menu bar was visible, the pull-down menu is also drawn. The first time a menu bar is drawn, no pull-down menu is displayed.
- The menu bar and the pull-down menu are drawn in front of all the applications windows.
- Screen regions obscured by the menus are saved by this function and restored by [MenuEraseStatus](#).
- See Also** [MenuInit](#), [MenuEraseStatus](#), [MenuDispose](#)

## MenuEraseStatus

**Purpose** Erase the menu command status.

**Prototype** `void MenuEraseStatus (MenuBarPtr MenuP)`

**Parameters** MenuP                      Pointer to a MenuBarType data structure, or NULL for the current menu.

**Result** Returns nothing.

**Comments** 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 before executing a menu command that displays a new form or one that causes the screen display to be modified.

When a menu command displays a new form, the system may save the bits in the current form. In some cases, the saved bits include the menu status; when you return to the original form later, the saved menu status is displayed and not erased. To avoid this problem, call `MenuEraseStatus` before executing the menu command that displays the new form.

You should also call this function before issuing a menu command which causes the bits under the menu status to be modified. For example, you need to call `MenuEraseStatus` before using menu commands to update text or other items. However, you need not do so for static buttons under the menu status (which is the most common case) because their appearance does not change.

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

## **MenuGetActiveMenu**

**Purpose** Returns a pointer to the current menu.

**Prototype** MenuBarPtr MenuGetActiveMenu (void)

**Parameters** None.

**Result** Returns a pointer to the current menu, NULL if there is none.

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

## MenuHandleEvent

**Purpose** Handle events in the current menu. This routine handles two types of events, [penDownEvent](#) and [winEnterEvent](#).

**Prototype**

```
Boolean MenuHandleEvent ( MenuBarPtr MenuP,  
                          EventPtr event,  
                          WordPtr error)
```

**Parameters**

MenuP	Pointer to a MenuBarType data structure.
event	Pointer to an EventType structure.
error	Error (or 0 if no error).

**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.

**Comments** When MenuHandleEvent receives a [penDownEvent](#), it checks if the pen position is within the bounds of the menu object. If it is, MenuHandleEvent tracks the pen until it comes up. If the pen comes up within the bounds of the menu, a [winEnterEvent](#) is added to the event queue, and the routine is exited.

When MenuHandleEvent receives a [winEnterEvent](#), it checks that the menu ID in the event record matches the ID of the specified menu. If there is a match, MenuHandleEvent creates and displays a popup window containing the menu's choices, and the routine is exited.

If a [penDownEvent](#) is received while the menu's popup window is displayed and the pen position is outside the bounds of the popup window, the menu is dismissed. If the pen position is within the bounds of the window, MenuHandleEvent tracks the pen until it comes up. If the pen comes up in the menu, a [winExitEvent](#) is added to the event queue.

## MenuInit

- Purpose** Load a menu resource from a resource file.
- Prototype** `MenuBarPtr MenuInit (Word resourceId)`
- Parameters** `resourceId` ID that identifies the menu resource.
- Result** Returns the pointer to a memory block allocated to hold the menu resource (a pointer to a `MenuBarType` data structure).
- Comments** The menu is not usable until [MenuSetActiveMenu](#) is called.
- See Also** [MenuSetActiveMenu](#), [MenuDispose](#)

## MenuSetActiveMenu

- Purpose** Set the current menu.
- Prototype** `MenuBarPtr MenuSetActiveMenu (MenuBarPtr MenuP)`
- Parameters** `MenuP` Pointer to the memory block that contains the new menu, or NULL for none.
- Result** Returns a pointer to the menu that was active before the new menu was set, or NULL if no menu was active.
- See Also** [MenuGetActiveMenu](#)

## Miscellaneous User Interface Functions

### AbtShowAbout

**Purpose** Displays the info dialog box. The application name is picked up from either the application name resource, defined in constructor, or the name of the application database (which is assigned in the makefile).

**Prototype** `void AbtShowAbout (ULong creator)`

**Parameters** `creator` Creator ID of this application.

**Result** Returns nothing.

### DayHandleEvent

**Purpose** Handle event in the specified control. This routine handles two type of events, [penDownEvent](#) and [ctlEnterEvent](#).

**Prototype** `Boolean DayHandleEvent ( DaySelectorPtr pSelector,  
EventPtr pEvent )`

**Parameters** `pSelector` Pointer to control object (ControlType)  
`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.  
A date is used if the user selects a day in the visible month.

## KeySetMask

**Purpose** Specify which keys generate `keyDownEvents`.  
You can specify this either by using this function or by using the `powerOn` modifier.

**Prototype** `DWord KeySetMask (DWord keyMask)`

**Parameters** `keyMask` Mask with bits set for those keys to generate `keyDownEvents` for.

**Result** Returns the old `keyMask`.

## LocGetNumberSeparators

**Purpose** Get localized number separators.

**Prototype** `void LocGetNumberSeparators(  
NumberFormatType numberFormat,  
Char *thousandSeparator,  
Char *decimalSeparator)`

**Parameters** `numberFormat` The format to use  
`thousandSeparator` Return a localized thousand separator here (allocate 1 char).  
`decimalSeparator` Return a localized decimal separator here (allocate 1 char).

**Result** Returns nothing.

**See Also** `StrLocalizeNumber`, `StrDelocalizeNumber` (documented in “Developing Palm OS Applications, Part II)

## Pen Manager Functions

### PenCalibrate

**Purpose** Set the calibration of the pen.

**Prototype** `Err PenCalibrate (    PointType* digTopLeftP,  
                          PointType* digBotRightP,  
                          PointType* scrTopLeftP,  
                          PointType* scrBotRightP)`

**Parameters**

<code>digTopLeftP</code>	Digitizer output from top-left coordinate.
<code>digBotRightP</code>	Digitizer output from bottom-right coordinate.
<code>scrTopLeftP</code>	Screen coordinate near top-left corner.
<code>scrBotRightP</code>	Screen coordinate near bottom-right corner.

**Result** Returns 0 if no error.

**Comments** Called by Preferences application when calibrating pen.

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

## PenResetCalibration

- Purpose** Reset the calibration in preparation for calibrating the pen again.
- 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!

---

## Functions for System Use Only

### PenClose

- Prototype** `Err PenClose (void)`

---

WARNING: This function for use by system software only.

---

### PenGetRawPen

- Prototype** `Err PenGetRawPen (PointType* penP)`
- See Instead** `EvtDequeuePenPoint` (documented in “*Developing Palm OS Applications, Part II*”)

---

WARNING: This function for use by system software only.

---

## Palm OS User Interface Functions

### *Pen Manager Functions*

---

#### **PenOpen**

**Prototype** `Err PenOpen (void)`

---

WARNING: This function for use by system software only.

---

#### **PenSleep**

**Prototype** `Err PenSleep (void)`

---

WARNING: This function for use by system software only.

---

#### **PenRawToScreen**

**Prototype** `Err PenRawToScreen (PointType* penP)`

---

WARNING: This function for use by system software only.

---

#### **PenScreenToRaw**

**Prototype** `Err PenScreenToRaw (PointType* penP)`

---

WARNING: This function for use by system software only.

---

#### **PenWake**

**Prototype** `Err PenWake (void)`

---

WARNING: This function for use by system software only.

---

## Progress Manager Functions

### PrgHandleEvent

<b>Purpose</b>	Handles events related to the active progress dialog.				
<b>Prototype</b>	<pre>Boolean PrgHandleEvent ( ProgressPtr prgP,                         EventPtr eventP)</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; eventP</td><td>Pointer to an event. You can pass a NULL event to cause this function to immediately call your <code>textCallback</code> function and then update the dialog (for example, after you call <code>PrgUpdateDialog</code>).</td></tr></table>	--> prgP	Pointer to a progress structure created by <a href="#">PrgStartDialog</a> .	--> eventP	Pointer to an event. You can pass a NULL event to cause this function to immediately call your <code>textCallback</code> function and then update the dialog (for example, after you call <code>PrgUpdateDialog</code> ).
--> prgP	Pointer to a progress structure created by <a href="#">PrgStartDialog</a> .				
--> eventP	Pointer to an event. You can pass a NULL event to cause this function to immediately call your <code>textCallback</code> function and then update the dialog (for example, after you call <code>PrgUpdateDialog</code> ).				
<b>Result</b>	Returns <code>TRUE</code> if the system handled the event. If it returns <code>FALSE</code> , you should check if the user canceled the dialog by calling <a href="#">PrgUserCancel</a> .				
<b>Comments</b>	<p>Use this function instead of <code>SysHandleEvent</code> when you have a progress dialog. <code>PrgHandleEvent</code> internally calls <code>SysHandleEvent</code> 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 <code>appStopEvent</code> 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 <code>textCallback</code> function you specified in your call to <a href="#">PrgStartDialog</a> is called. Your <code>textCallback</code> function should set the <code>textP</code> buffer in the <code>PrgCallbackData</code> structure with the new message to be displayed in the progress dialog. Optionally, you can also set the <code>bitmapId</code> field to include an icon in the update</p>				

## Palm OS User Interface Functions

### Progress Manager Functions

---

dialog. For more information about the `textCallback` function, see the section “Progress `textCallback` Function” on page 186.

**See Also** [PrgStartDialog](#), [PrgStopDialog](#), [PrgUpdateDialog](#), [PrgUserCancel](#)

## **PrgStartDialog**

**Purpose** Displays a progress dialog that can be updated.

**Prototype**

```
ProgressPtr PrgStartDialog ( CharPtr 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. For more information about this function, see the section “Progress `textCallback` Function” on page 186.

**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.

**See Also** [PrgHandleEvent](#), [PrgStopDialog](#), [PrgUpdateDialog](#), [PrgUserCancel](#)

## **PrgStopDialog**

**Purpose** Releases memory used by the progress dialog and restores the screen to its initial state.

**Prototype** `void PrgStopDialog ( ProgressPtr prgP,  
                          Boolean force)`

**Parameters**

--> 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.

**Result** Returns nothing.

**Comments** If the progress dialog is in a state where it is displaying an error message to the user, this function normally waits for the user to confirm the dialog before it removes the dialog. If you specify TRUE for the `force` parameter, this causes the system to remove the dialog immediately.

**See Also** [PrgHandleEvent](#), [PrgStartDialog](#), [PrgUpdateDialog](#), [PrgUserCancel](#)

## **PrgUpdateDialog**

**Purpose** Updates the status of the current progress dialog.

**Prototype** `void PrgUpdateDialog (ProgressPtr prgP,  
Word err,  
Word stage,  
CharPtr messageP,  
Boolean updateNow)`

**Parameters**

--> prgP	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 “Progress textCallback Function” on page 186.

**See Also** [PrgHandleEvent](#), [PrgStartDialog](#), [PrgStopDialog](#), [PrgUserCancel](#)

## **PrgUserCancel**

- Purpose** Returns TRUE if the user cancelled the process via the progress dialog.
- Prototype** Word PrgUserCancel (ProgressPtr prgP)
- Parameters** --> prgP                    Pointer to a progress structure created by [PrgStartDialog](#).
- Result** Returns the value of the `cancel` field in the progress structure.
- Comments** This is a macro you can use to check if the user cancelled the process. If the user did cancel, you can change the progress dialog text to something like “Cancelling,” or “Disconnecting,” or whatever is appropriate for your application. Then you should cancel the process, end the communication session, or do whatever processing is necessary.
- See Also** [PrgHandleEvent](#), [PrgStartDialog](#), [PrgStopDialog](#), [PrgUpdateDialog](#)

## Rectangle Functions

### RctCopyRectangle

**Purpose** Copy the source rectangle to the destination rectangle.

**Prototype** `void RctCopyRectangle (const RectanglePtr srcRect,  
const RectanglePtr dstRect)`

**Parameters** `srcRect` A pointer to the rectangle to be copied.  
`dstRect` A pointer to the destination rectangle.

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

### RctGetIntersection

**Purpose** Determine the intersection of two rectangles.

**Prototype** `void RctGetIntersection ( const RectanglePtr r1,  
const RectanglePtr r2,  
RectanglePtr r3)`

**Parameters** `r1` A pointer to a source rectangle.  
`r2` A pointer to the other source rectangle.  
`r3` 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 rectangles `r1` and `r2` do not intersect, `r3` contains a rectangle that begins at coordinates (0, 0) and has 0 width and 0 height.

## RctInsetRectangle

**Purpose** Move all of the boundaries of a rectangle by a specified offset.

**Prototype** `void RctInsetRectangle ( const RectanglePtr r,  
SWord insetAmt)`

**Parameters**

<code>r</code>	A pointer to the rectangle.
<code>insetAmt</code>	Number of pixels to move the boundaries. This can be a negative number.

**Comments** The rectangle type `RectangleType`, which is pointed to by `RectanglePtr`, stores the coordinates for the top-left corner of the rectangle plus the rectangle's width and height. This function adds `insetAmt` to the x and y values of the top-left coordinate and then adjusts the width and the height accordingly so that all of the sides of the rectangle are contracted or expanded by the same amount.

A positive `insetAmt` creates a smaller rectangle that is contained inside the old rectangle's boundaries. A negative `insetAmt` creates a larger rectangle that surrounds the old rectangle.

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

## **RctOffsetRectangle**

**Purpose** Move the top and left boundaries of a rectangle by the specified values.

**Prototype** `void RctOffsetRectangle (     const RectanglePtr r,  
                                  const SWord deltaX,  
                                  const SWord deltaY)`

**Parameters**

<code>r</code>	A pointer to the rectangle.
<code>deltaX</code>	Number of pixels to move the left boundary. This can be a negative number.
<code>deltaY</code>	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.

**Prototype**

```
Boolean RctPtInRectangle (  const SWord x,
                           const SWord y,
                           const RectanglePtr r)
```

**Parameters**

x	The x coordinate of the point.
y	The y coordinate of the point.
r	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.

**Prototype**

```
void RctSetRectangle (  const RectanglePtr r,
                        const SWord left,
                        const SWord top,
                        const SWord width,
                        const SWord height)
```

**Parameters**

r	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](#)

## Scrollbar Functions

### **ScIDrawScrollBar**

**Purpose** Draw a scroll bar.

**Prototype** `void ScIDrawScrollBar (ScrollBarPtr bar)`

**Parameters** `bar` Pointer to a scroll bar structure.

**Result** Returns nothing.

### **ScIGetScrollBar**

**Purpose** Retrieve a scrollbar's current position, its range, and the size of a page. If the scroll bar is visible, it's redrawn.

**Prototype** `void ScIGetScrollBar ( ScrollBarPtr bar,  
ShortPtr valueP,  
ShortPtr minP,  
ShortPtr maxP,  
ShortPtr pageSizeP)`

**Parameters** `bar` Pointer to a scroll bar structure.  
`valueP` Pointer to current value (position).  
`minP` Pointer to minimum value.  
`maxP` Pointer to maximum value.  
`pageSizeP` Pointer to size of a page (used when page scrolling).

**Result:** Returns nothing.  
Stores the current values in `valueP`, `minP`, `maxP`, and `pageSizeP`.

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

## ScIHandleEvent

<b>Purpose</b>	Handles the three scrollbar events.				
<b>Prototype</b>	<pre>Boolean ScIHandleEvent ( ScrollBarPtr bar,                         EventPtr event )</pre>				
<b>Parameters</b>	<table><tr><td>bar</td><td>Pointer to a scroll bar structure.</td></tr><tr><td>event</td><td>Pointer to an event (EventType data structure).</td></tr></table>	bar	Pointer to a scroll bar structure.	event	Pointer to an event (EventType data structure).
bar	Pointer to a scroll bar structure.				
event	Pointer to an event (EventType data structure).				
<b>Result</b>	Returns TRUE if the event was handled.				
<b>Comment</b>	<p>When the user touches a scroll bar with a pen, the system sends a <a href="#">sclEnterEvent</a>. Generally, applications don't need to respond to that event.</p> <p>When the user holds and drags the scroll bar with the pen, the system sends a <a href="#">sclRepeatEvent</a>. Applications that implement dynamic scrolling should catch this event and move the text each time one arrives.</p> <p>When the user releases the pen from the scroll bar, the system sends a <a href="#">sclExitEvent</a>. Applications that implement non-dynamic scrolling should catch this event and move the text when <code>sclExitEvent</code> arrives. Applications that implement dynamic scrolling can ignore this event.</p>				

## **ScISetScrollbar**

**Purpose** Set the scrollbar's current position, its range, and the size of a page. If the scroll bar is visible, it's redrawn.

**Prototype** `void ScISetScrollbar ( ScrollBarPtr bar,  
Short value,  
Short min,  
Short max,  
Short pageSize)`

<b>Parameters</b>	<code>bar</code>	Pointer to a scroll bar structure.
	<code>value</code>	Current value (position); one of the initialization values
	<code>min</code>	Minimum value.
	<code>max</code>	Maximum value.
	<code>pageSize</code>	Size of a page (used when page scrolling).

**Result** Returns nothing.

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



## Palm OS User Interface Functions

### *Functions for System Use Only*

---

#### **FindStrInStr**

**Prototype**    `Boolean FindStrInStr ( CharPtr strToSearch,  
                          CharPtr strToFind,  
                          WordPtr posP)`

---

WARNING: System Use Only!

---

#### **UIInitialize**

**Prototype**    `void UIInitialize (void)`

---

WARNING: System Use Only!

---

#### **UIReset**

**Prototype**    `void UIReset (void)`

---

WARNING: System Use Only!

---

## Time Selection Functions

### SelectDay

- Purpose** Display a form showing a date; allow user to select a different date.
- Prototype** `Boolean SelectDay ( SelectDayType selectDayBy,  
SWord *month,  
SWord *day,  
SWord *year,  
CharPtr title)`
- Parameters**
- |                               |                                |
|-------------------------------|--------------------------------|
| <code>selectDayBy</code>      |                                |
| <code>month, day, year</code> | Month, day, and year selected. |
| <code>title</code>            | String title for the dialog.   |
- Result** TRUE if the OK button was pressed. If TRUE, month, day, and year contain the new date.
- See Also** [SelectDayV10](#)

### SelectDayV10

- Purpose** Display a form showing a date, allow user to select a different date.
- Prototype** `Boolean SelectDay (int *month, int *day,  
int *year, CharPtr title)`
- Parameters**
- |                               |                               |
|-------------------------------|-------------------------------|
| <code>month, day, year</code> | Month, day and year selected. |
| <code>title</code>            | String title for the dialog.  |
- Result** Returns TRUE if the OK button was pressed. In that case, the parameters passed are changed.
- See Also** [SelectDay](#)

## SelectTime

**Purpose** Display a form showing the time and allow the user to select a different time.

**Prototype**

```
Boolean SelectTime ( TimePtr startTimeP,  
                    TimePtr endTimeP,  
                    Boolean untimed,  
                    CharPtr title,  
                    SWord startOfDay)
```

**Parameters**

<code>startTimeP, endTimeP</code>	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.
<code>untimed</code>	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 (-1)</code> .
<code>title</code>	A pointer to a string to display as the title. Doesn't change as the function executes.
<code>startOfDay</code>	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 expect the values that `startTimeP` and `endTimeP` are likely to be changed.

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

## Table Functions

### TblDrawTable

- Purpose** Draw a table.
- Prototype** `void TblDrawTable (TablePtr table)`
- Parameters** `table` Pointer to a table object.
- Result** Returns nothing.
- See Also** [TblEraseTable](#), [TblRedrawTable](#),  
[TblSetCustomDrawProcedure](#)

### TblEditing

- Purpose** Check whether a table is in edit mode.
- Prototype** `Boolean TblEditing (TablePtr table)`
- Parameters** `table` Pointer to a table object.
- Result** Returns `TRUE` if the table is in edit mode, `FALSE` otherwise.
- Comments** The table is in edit mode while the user edits a text item.

## **TblEraseTable**

**Purpose** Erase a table object.

**Prototype** `void TblEraseTable (TablePtr table)`

**Parameters** `table` Pointer to a table object.

**Result** Returns nothing.

**See Also** [TblDrawTable](#), [TblSetCustomDrawProcedure](#),  
[TblRedrawTable](#)

## **TblFindRowData**

**Purpose** Return the row number that contains the specified data value.

**Prototype** `Boolean TblFindRowData ( TablePtr table,  
                                  ULong data,  
                                  WordPtr rowP)`

**Parameters** `table` Pointer to a table object.  
`data` Row data to find.  
`rowP` Pointer to the row number (return value).

**Result** Returns TRUE if a match was found, FALSE otherwise.

**See Also** [TblGetRowData](#), [TblFindRowID](#)

## TblFindRowID

**Purpose** Return the number of the row that matches the specified ID.

**Prototype** `Boolean TblFindRowID ( TablePtr table,  
Word id,  
WordPtr rowP)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>id</code>	Row ID to find.
<code>rowP</code>	Pointer to the row number (return value).

**Result** Returns TRUE if a match was found, FALSE otherwise.

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

## TblGetBounds

**Purpose** Return the bounds of a table.

**Prototype** `void TblGetBounds (TablePtr table, RectanglePtr r)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>r</code>	Pointer to a RectangleType structure.

**Result** Returns nothing. Stores the bounds in `r`.

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

## **TblGetColumnSpacing**

**Purpose** Return the spacing after the specified column.

**Prototype** `Word TblGetColumnSpacing ( TablePtr table,  
Word column)`

**Parameters** `table` Pointer to a table object.  
`column` Column number (zero-based).

**Result** Returns the spacing after column (in pixels).

**See Also** [TblGetColumnWidth](#), [TblSetColumnSpacing](#),  
[TblSetColumnUsable](#)

## **TblGetColumnWidth**

**Purpose** Return the width of the specified column.

**Prototype** `Word TblGetColumnWidth ( TablePtr table,  
Word column)`

**Parameters** `table` Pointer to a table object.  
`column` Column number (zero-based).

**Result** Returns the width of a column (in pixels).

**See Also** [TblGetColumnSpacing](#), [TblSetColumnWidth](#),  
[TblSetColumnUsable](#)

## TblGetCurrentField

**Purpose** Return a pointer to the `FieldType` structure in which the user is currently editing a text item.

**Prototype** `FieldPtr TblGetCurrentField (TablePtr table)`

**Parameters** `table` Pointer to a table object.

**Result** Returns `FieldPtr`, or `NULL` if the table is not in edit mode.

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

## TblGetItemBounds

**Purpose** Return the bounds of an item in a table.

**Prototype** `void TblGetItemBounds ( TablePtr table,  
Word row,  
Word column,  
RectanglePtr r )`

**Parameters** `table` Pointer to a table object.  
`row` Row of the item (zero-based).  
`column` Column of the item (zero-based).  
`r` Pointer to a structure that holds the bounds of the item.

**Result** Returns nothing. Stores the bounds in `r`.

## **TblGetItemFont**

**Purpose** Return the font used to display a table item.

**Prototype** `FontID TblGetItemFont ( TablePtr table,  
Word row,  
Word column)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>row</code>	Row of the item to select (zero-based).
<code>column</code>	Column of the item to select (zero-based).

**Result** Returns the ID of the font used for the table item at the row and column indicated.

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

## **TblGetItemInt**

**Purpose** Return the integer value stored in a table item.

**Prototype** `Word TblGetItemInt ( TablePtr table,  
Word row,  
Word column)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>row</code>	Row of the item to select (zero-based).
<code>column</code>	Column of the item to select (zero-based).

**Result** Returns the integer value.

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

## TblGetLastUsableRow

- Purpose** Return the last row in a table that is usable (visible).
- Prototype** `Word TblGetLastUsableRow (TablePtr table)`
- Parameters** `table` Pointer to a table object.
- Result** Returns the row index (zero-based) or -1 if there are no usable rows.
- See Also** [TblGetRowData](#), [TblGetRowID](#)

## TblGetNumberOfRows

- Purpose** Return the number of rows in a table.
- Prototype** `Word TblGetNumberOfRows (TablePtr table)`
- Parameters** `table` Pointer to a table object.
- Result** Returns the number of rows in the specified table.

## TblGetRowData

- Purpose** Return the data value of the specified row. The data value is a placeholder for application-specific values.
- Prototype** `ULong TblGetRowData (TablePtr table, Word row)`
- Parameters** `table` Pointer to a table object.  
`row` Row of the item to select (zero-based).
- See Also** [TblGetRowID](#), [TblSetRowData](#)

## **TblGetRowHeight**

**Purpose** Return the height of the specified row.

**Prototype** `Word TblGetRowHeight (TablePtr table, Word row)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>row</code>	Row to get (zero-based).

**Result** Returns the height in pixels.

**See Also** [TblGetItemBounds](#), [TblSetRowHeight](#)

## **TblGetRowID**

**Purpose** Return the ID value of the specified row.

**Prototype** `Word TblGetRowID (TablePtr table, Word row)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>row</code>	Row for which the ID will be returned (zero-based).

**Result** Returns the ID value of the row in the table.

**See Also** [TblGetRowData](#), [TblSetRowHeight](#)

## TblGetSelection

- Purpose** Return the row and column of the currently selected table item.
- Prototype**

```
Boolean TblGetSelection ( TablePtr table,  
                          WordPtr rowP,  
                          WordPtr columnP)
```
- Parameters**
- |               |  |
|---------------|--|
| table         | Pointer to a table object.   |
| rowP, columnP | Pointer to a Word variable in which to store the row /column (zero-based). |
- Result** Returns TRUE if the item is highlighted, FALSE if not.
- See Also** [TblSetRowSelectable](#)

## TblGrabFocus

- Purpose** Put a table into edit mode.
- Prototype**

```
void TblGrabFocus ( TablePtr table,  
                   Word row,  
                   Word column)
```
- Parameters**
- |        |   |
|--------|---|
| table  | Pointer to a table object.                |
| row    | Current row to be edited (zero-based).    |
| column | Current column to be edited (zero-based). |
- Result** Returns nothing.
- Comments** Displays an error if the row or column passed is out of bounds. An editable field must exist in the coordinates passed to this function.
- See Also** [TblReleaseFocus](#)

## **TblHandleEvent**

**Purpose** Handle an event for the table.

**Prototype** `Boolean TblHandleEvent ( TablePtr table,  
EventPtr event )`

**Parameters** `table` Pointer to a table object.  
`event` The event to be handled.

**Result** Returns `TRUE` if the event was handled, `FALSE` if it was not.

## **TblHasScrollBar**

**Purpose** Set the `hasScrollBar` attribute in the table. A table that has its attribute set will initialize the associated field object such that it will send `fldChanged` events when that scroll bar needs to be updated.

**Prototype** `void TblHasScrollBar ( TablePtr table,  
Boolean hasScrollBar )`

**Parameters** `table` Pointer to a table object  
`hasScrollBar` `TRUE` to set the attribute, `FALSE` to unset it.

**Result** Returns nothing.

## TblInsertRow

- Purpose** Insert a row into the table before the specified row.  
The number of rows in the table is not increased; the last row in the table is removed.
- Prototype** `void TblInsertRow (TablePtr table, Word row)`
- Parameters**
- |                    |                             |
|--------------------|-----------------------------|
| <code>table</code> | Pointer to a table object.  |
| <code>row</code>   | Row to insert (zero-based). |
- Result** Returns nothing.
- Comments** If the `row` parameter is greater than or equal to the number of rows in the table, an error is displayed.
- See Also** [TblRemoveRow](#), [TblSetRowUsable](#), [TblSetRowSelectable](#), [TblMarkRowInvalid](#)

## TblMarkRowInvalid

- Purpose** Mark the image of the specified row invalid.
- Prototype** `void TblMarkRowInvalid (TablePtr table, Word row)`
- Parameters**
- |                    |   |
|--------------------|---|
| <code>table</code> | Pointer to a table object.              |
| <code>row</code>   | Row of the item to select (zero-based). |
- Comments** After calling this function, call [TblRedrawTable](#) to redraw all rows marked invalid. Rows not marked invalid are not redrawn.
- Result** Returns nothing.
- See Also** [TblRemoveRow](#), [TblSetRowUsable](#), [TblSetRowSelectable](#), [TblMarkTableInvalid](#), [TblRowInvalid](#)

## **TblMarkTableInvalid**

**Purpose** Mark the image of all the rows in a table invalid.

**Prototype** `void TblMarkTableInvalid (TablePtr table)`

**Parameters** `table` Pointer to a table object.

**Result** Returns nothing.

**Comments** After calling this function, you must call [TblRedrawTable](#) to redraw all rows. Rows not marked invalid are not redrawn.

**See Also** [TblEraseTable](#), [TblRedrawTable](#)

## **TblRedrawTable**

**Purpose** Redraw the rows of the table that are marked invalid.

**Prototype** `void TblRedrawTable (TablePtr table)`

**Parameters** `table` Pointer to a table object.

**Result** Returns nothing.

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

## TblReleaseFocus

- Purpose** Release the focus.
- Prototype** `void TblReleaseFocus (TablePtr table)`
- Parameters** `table` Pointer to a table object.
- Result** Returns nothing.
- Comments** If the current item is a text item, the memory allocated for editing is released and the insertion point is turned off.
- See Also** [TblGrabFocus](#)

## TblRemoveRow

- Purpose** Remove the specified row from the table.
- Prototype** `void TblRemoveRow (TablePtr table, Word row)`
- Parameters** `table` Pointer to a table object.  
`row` Row to remove (zero-based).
- Result** Returns nothing.
- Comments** The number of rows in the table is not decreased; an unusable row is added to the end of the table. If an invalid row is specified, an error is displayed.  
This function does not visually update the display.
- See Also** [TblInsertRow](#), [TblSetRowUsable](#), [TblSetRowSelectable](#), [TblMarkRowInvalid](#)

## **TblRowInvalid**

**Purpose** Determine whether a row is invalid. Invalid rows need to be redrawn.

**Prototype** `Boolean TblRowInvalid (TablePtr table, Word row)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>row</code>	Row number (zero-based).

**Result** Returns `TRUE` if the row is invalid, `FALSE` if it's valid.

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

## **TblRowSelectable**

**Purpose** Determine whether the specified row is selectable. Rows that are not selectable don't highlight when touched.

**Prototype** `Boolean TblRowSelectable ( TablePtr table,  
Word row)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>row</code>	Row of the item to select (zero-based).

**Result** Returns `TRUE` if the row is selectable, `FALSE` if it's not.

## TblRowUsable

- Purpose** Determine whether the specified row is usable.
- Prototype** `Boolean TblRowUsable (TablePtr table, Word row)`
- Parameters**
- |                    |                            |
|--------------------|----------------------------|
| <code>table</code> | Pointer to a table object. |
| <code>row</code>   | Row number (zero-based).   |
- Result** Returns `TRUE` if the row is usable, `FALSE` if it's not.
- Comments** Rows that are not usable do not display.
- See Also** [TblRowSelectable](#), [TblGetLastUsableRow](#)

## TblSelectItem

- Purpose** Select (highlight) the specified item. If there is already a selected item, it is unhighlighted.
- Prototype** `void TblSelectItem ( TablePtr table,  
Word row,  
Word column)`
- Parameters**
- |                     |  |
|---------------------|--|
| <code>table</code>  | Pointer to a table object.                 |
| <code>row</code>    | Row of the item to select (zero-based).    |
| <code>column</code> | Column of the item to select (zero-based). |
- Result** Returns nothing.
- See Also** [TblRowSelectable](#), [TblGetItemBounds](#), [TblGetItemInt](#)

## **TblSetBounds**

**Purpose** Sets the bounds of a table.

**Prototype** `void TblSetBounds (TablePtr table, RectanglePtr r)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>r</code>	Pointer to a <code>RectangleType</code> structure that specifies the bounds for the table.

**Result** Returns nothing.

## **TblSetColumnEditIndicator**

**Purpose** Set the column attribute that controls whether a column highlights when the table is in edit mode.

**Prototype** `void TblSetColumnEditIndicator (`  
`TablePtr table,`  
`Word column,`  
`Boolean editIndicator)`

**Parameters**

<code>table</code>	Pointer to a table object
<code>column</code>	Column of the item (zero based)
<code>editIndicator</code>	TRUE to highlight, FALSE to turn off highlight.

**Result** Returns nothing.

## TblSetColumnSpacing

- Purpose** Set the spacing after the specified column.
- Prototype**

```
void TblSetColumnSpacing ( TablePtr table,  
                          Word column,  
                          Word spacing)
```
- Parameters**
- |         |                             |
|---------|-----------------------------|
| table   | Pointer to a table object.  |
| column  | Column number (zero-based). |
| spacing | Spacing after the column.   |
- Result** Returns nothing.
- See Also** [TblSetColumnUsable](#)

## TblSetColumnUsable

- Purpose** Set a column in a table usable or unusable.
- Prototype**

```
void TblSetColumnUsable ( TablePtr table,  
                          Word row,  
                          Boolean usable)
```
- Parameters**
- |        |  |
|--------|--|
| table  | Pointer to a table object.               |
| row    | Row of the item to select (zero-based).  |
| usable | TRUE for usable or FALSE for not usable. |
- Result** Returns nothing.
- Comments** Columns that are not usable do not display.
- See Also** [TblMarkRowInvalid](#)

## **TblSetColumnWidth**

**Purpose** Set the width of the specified column.

**Prototype** `void TblSetColumnWidth ( TablePtr table,  
Word column,  
Word width)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>column</code>	Column number (zero-based).
<code>width</code>	Width of the column (in pixels).

**Result** Returns nothing.

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

## TblSetCustomDrawProcedure

**Purpose** Set the custom draw callback procedure for the specified column.

**Prototype**

```
void TblSetCustomDrawProcedure (TablePtr table,  
                                Word column,  
                                VoidPtr drawCallback)
```

**Parameters**

table	Pointer to a table object.
column	Column of table.
drawCallback	Callback function.

**Result** Returns nothing.

**Comments** The custom draw callback function is used to draw table items with a `TableItemStyleType` of `customTableItem` (see `table.h`).

The callback procedure should have this prototype:

```
void TableDrawItemFuncType (  
    VoidPtr table,  
    Word row,  
    Word column,  
    RectanglePtr bounds);
```

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

### **TblSetItemFont**

**Purpose** Set the font used to display a table item.

**Prototype** `void TblSetItemFont ( TablePtr table,  
Word row,  
Word column,  
FontID fontID)`

**Parameters**

table	Pointer to a table object.
row	Row of the item to select (zero-based).
column	Column of the item to select (zero-based).
fontID	ID of the font to be used.

**Result** Returns nothing.

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

### **TblSetItemInt**

**Purpose** Set the integer value of the specified item.

**Prototype** `void TblSetItemInt ( TablePtr table,  
Word row,  
Word column,  
Word value)`

**Parameters**

table	Pointer to a table object.
row	Row of the item (zero-based).
column	Column of the item (zero-based).
value	Any byte value (an integer).

**Result** Returns nothing.

**Comments** An application can store what it wants in an item's integer value.

**See Also** [TblGetItemInt](#), [TblSetItemPtr](#)

## TblSetItemPtr

**Purpose** Set the item to the specified pointer value.

**Prototype**

```
void TblSetItemPtr (   TablePtr table,
                      Word row,
                      Word column,
                      VoidPtr value)
```

**Parameters**

table	Pointer to a table object.
row	Row of the item (zero-based).
column	Column of the item (zero-based).
value	Pointer to data to display in the table item.

**Result** Returns nothing.

**Comments** An application can store whatever it wants in the table item.

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

## **TblSetItemStyle**

**Purpose** Set the item to display its data in a style; for example, text, numbers, dates, and so on.

**Prototype**

```
void TblSetItemStyle ( TablePtr table,  
                      Word row,  
                      Word column,  
                      TableItemStyleType type)
```

**Parameters**

table	Pointer to a table object.
row	Row of the item (zero-based).
column	Column of the item (zero-based).
type	See <code>Table.h</code> .

**Result** Returns nothing.

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

## TblSetLoadDataProcedure

**Purpose** Set the load-data callback procedure for the specified column.

**Prototype**

```
void TblSetLoadDataProcedure (  
    TablePtr table,  
    Word column,  
    TableLoadDataFuncPtr loadDataCallback)
```

**Parameters**

table	Pointer to a table object.
column	Column of table.
loadDataCallback	Callback procedure.

**Result** Returns nothing.

**Comments** The callback procedure is used to obtain the data values of a table item. It should have this prototype:

```
Err TableLoadDataFuncType (  
    VoidPtr table,  
    Word row,  
    Word column,  
    Boolean editable,  
    VoidHand *dataH,  
    WordPtr dataOffset,  
    WordPtr dataSize,  
    FieldPtr fld)
```

For a text style item, the callback procedure should return the handle of a block that contains a null-terminated text string, the offset from the start of the block to the start of the string, and the amount of space allocated for the string.

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

## **TblSetRowData**

**Purpose** Set the data value of the specified row.  
The data value is a placeholder for application-specific values.

**Prototype** `void TblSetRowData ( TablePtr table,  
Word row,  
ULong data)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>row</code>	Row of the item to select (zero-based).
<code>data</code>	Application-specific data.

**Result** Returns nothing.

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

## **TblSetRowHeight**

**Purpose** Set the height of the specified row.

**Prototype** `void TblSetRowHeight ( TablePtr table,  
Word row,  
Word height)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>row</code>	Row to set (zero-based).
<code>height</code>	New height in pixels.

**Result** Returns nothing.

**See Also** [TblGetRowHeight](#), [TblSetRowStaticHeight](#)

## TblSetRowID

- Purpose** Set the ID value of the specified row.
- Prototype**

```
void TblSetRowID ( TablePtr table,  
                  Word row,  
                  Word id)
```
- Parameters**
- |       |   |
|-------|---|
| table | Pointer to a table object.              |
| row   | Row of the item to select (zero-based). |
| id    | ID to identify a row.                   |
- Result** Returns nothing.
- See Also** [TblGetRowID](#)

## TblSetRowSelectable

- Purpose** Set a row in a table to selectable or nonselectable.
- Prototype**

```
void TblSetRowSelectable ( TablePtr table,  
                           Word row,  
                           Boolean selectable)
```
- Parameters**
- |            |   |
|------------|---|
| table      | Pointer to a table object.              |
| row        | Row of the item to select (zero-based). |
| selectable | TRUE or FALSE.                          |
- Result** Returns nothing.
- Comments** Rows that are not selectable don't highlight when touched.
- See Also** [TblRowSelectable](#), [TblSetRowUsable](#)

## TblSetRowStaticHeight

**Purpose** Set the static height attribute of a row. 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.

**Prototype** `void TblSetRowStaticHeight ( TablePtr table,  
Word row,  
Boolean staticHeight)`

**Parameters**

<code>table</code>	Pointer to a table object
<code>row</code>	Row of the item to select (zero based)
<code>staticHeight</code>	TRUE to set the static height, FALSE to unset it.

**Result** Nothing.

## TblSetRowUsable

**Purpose** Set a row in a table to usable or unusable. Rows that are not usable do not display.

**Prototype** `void TblSetRowUsable ( TablePtr table,  
Word row,  
Boolean usable)`

**Parameters**

<code>table</code>	Pointer to a table object.
<code>row</code>	Row of the item to select (zero-based).
<code>usable</code>	TRUE or FALSE.

**Result** Returns nothing.

**See Also** [TblRowUsable](#), [TblSetRowSelectable](#)

## TblSetSaveDataProcedure

**Purpose** Set the save-data callback procedure for the specified column.

**Prototype**

```
void TblSetSaveDataProcedure (TablePtr table,  
                             Word column,  
                             VoidPtr saveDataCallback)
```

**Parameters**

table	Pointer to a table object.
column	Column of table.
saveDataCallback	Callback function.

**Comments** The callback procedure is called when the table object determines the data of a text object needs to be saved.

The callback procedure should have this prototype:

```
Boolean TableSaveDataFuncType  
(VoidPtr table,  
 Word row,  
 Word column);
```

**Result** Returns nothing.

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

## TblUnhighlightSelection

**Purpose** Unhighlight the currently selected item in a table.

**Prototype**

```
void TblUnhighlightSelection (TablePtr table)
```

**Parameters**

table	Pointer to a table object.
-------	----------------------------

**Result** Returns nothing.



`scrDisplayModeGetSupportsColor`  
Return TRUE as the value of the `enableColorP` parameter when color mode can be enabled. On Palm III devices, this operation always returns FALSE.

`scrDisplayModeSet`  
Change display settings to the values specified by the other arguments to the `ScrDisplayMode` function.

`scrDisplayModeSetToDefaults`  
Change display settings to default values. See [Comments](#) at the end of this description for values.

<code>widthP</code>	Pointer to new/old screen width. On Palm III devices this value must always be NULL or a pointer to a DWord that contains the value 160.
<code>heightP</code>	Pointer to new/old screen height. On Palm III devices this value must always be NULL or a pointer to a DWord that contains the value 160.
<code>depthP</code>	Pointer to new/old/available screen depth. On Palm III devices this value must always be NULL or a pointer to a DWord that contains either of the values 1 or 2.
<code>enableColorP</code>	Pass TRUE to enable color drawing mode. On Palm III devices, this value must always be FALSE or NULL.

**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 `scrDisplayModeSet` operation changes current display

## Palm OS User Interface Functions

### Window Functions

---

parameters when passed valid argument values that are not `NULL` pointers. The `scrDisplayModeSetToDefaults` operation ignores values passed for all of these parameters.

Table 7.1 summarizes parameter usage for each operation this function performs.

**Table 7.1 Use of parameters to `ScrDisplayMode` function**

Operation <code>scrDisplayMode...</code>	<code>widthP</code>	<code>heightP</code>	<code>depthP</code>	<code>enableColorP</code>
<code>...Get</code>	returned	returned	returned	returned
<code>...GetDefaults</code>	returned	returned	returned	returned
<code>...GetSupportedDepths</code>	pass in	pass in	returned	pass in
<code>...GetSupportsColor</code>	pass in	pass in	pass in	returned
<code>...Set</code>	pass in	pass in	pass in	pass in
<code>...SetToDefaults</code>	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.

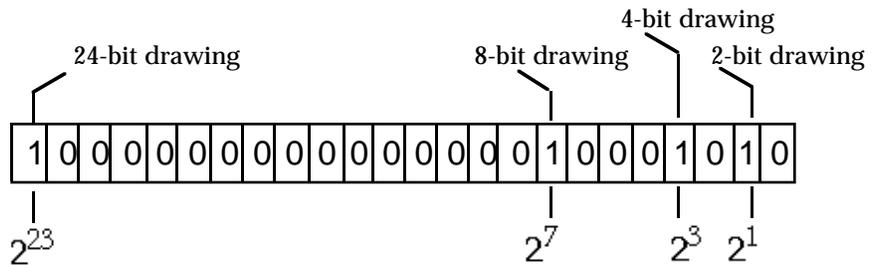
Default display values for Palm OS 3.0 running on Palm III hardware are

```
width: 160,  
height: 160,  
depth: FtrGet(sysFtrNumDisplayDepth),  
enableColor: FALSE
```

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.

**Example** Here are some additional examples of return values provided by the `scrDisplayModeGetSupportedDepths` mode of the `ScrDisplayMode` 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 24, 8, 4 and 2 is indicated by `0x80008A`. The figure immediately following depicts this final example graphically.



Bit depth support indicated by interpreting `0x80008A` as binary value

## **WinAddWindow**

**Purpose** Add the specified window to the active windows list.

**Prototype** `void WinAddWindow (WinHandle winHandle)`

**Parameters** `winHandle` Handle of a window.

**Result** Returns nothing.

**Comment** The active windows list contains all windows in the current application's user interface.

**See Also** [WinCreateWindow](#), [WinRemoveWindow](#)

## **WinClipRectangle**

**Purpose** Clip a rectangle to the clipping rectangle of the draw window.

**Prototype** `void WinClipRectangle (RectanglePtr r)`

**Parameters** `r` Pointer to a structure holding the rectangle to clip.

**Comment** The draw window is the window to which all drawing functions send their output.

The rectangle returned in `r` is the intersection of the rectangle passed and the draw window's clipping bounds.

**Result** Returns nothing.

**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).

**Prototype**

```
void WinCopyRectangle (WinHandle srcWin,  
                      WinHandle dstWin,  
                      RectanglePtr srcRect,  
                      SWord destX,  
                      SWord destY,  
                      ScrOperation mode)
```

<b>Parameters</b>	<code>srcWin</code>	Window from which the rectangle is copied.
	<code>dstWin</code>	Window to which the rectangle is copied.
	<code>srcRect</code>	Bounds of the region to copy.
	<code>destX</code>	Top bound of the rectangle in destination window.
	<code>destY</code>	Left bound of the rectangle in destination window.
	<code>mode</code>	The method of transfer from the source to the destination window (see <code>Window.h</code> ).

**Result** Returns nothing.

**Comments** Copies the bits of the window inside the rectangle region.

## **WinCreateOffscreenWindow**

**Purpose** Create a new off-screen window and add it to the window list.

**Prototype** `WinHandle WinCreateOffscreenWindow (  
    SWord width,  
    SWord height,  
    WindowFormatType format,  
    WordPtr error)`

**Parameters**

<code>width, height</code>	Width and height of the window in pixels.
<code>format</code>	Either <code>screenFormat</code> or <code>genericFormat</code> .
<code>error</code>	Pointer to any error this function encounters.

**Result** Returns the handle of the new window.

**Comments** Windows created with this routine draw to a memory buffer instead of the display.

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.

**See Also** [WinCreateWindow](#), [WinAddWindow](#)

## WinCreateWindow

**Purpose** Create a new window and add it to the window list.

**Prototype** WinHandle WinCreateWindow ( RectanglePtr bounds,  
FrameType frame,  
Boolean modal,  
Boolean focusable,  
WordPtr error)

<b>Parameters</b>	bounds	Display relative bounds of the window.
	frame	Type of frame around the window (see Window.h).
	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 a handle for the new window.

**Comments** Windows created by this routine draw to the display. See WinCreateOffscreenWindow for information on drawing off screen.

New windows are created disabled, and must be enabled before they accept input.

**See Also** [WinCreateOffscreenWindow](#), [WinDeleteWindow](#), [WinInitializeWindow](#)

## **WinDeleteWindow**

**Purpose** Remove a window from the window list and free the memory used by the window.

**Prototype** `void WinDeleteWindow ( WinHandle winHandle,  
Boolean eraseIt )`

**Parameters**

<code>winHandle</code>	Handle of window to delete.
<code>eraseIt</code>	If TRUE, the window is erased before it is deleted.

**Result** Returns nothing.

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

## **WinDisableWindow**

**Purpose** Disable a window but leave it on the active windows list (list of all windows in the system).

**Prototype** `void WinDisableWindow (WinHandle winHandle)`

**Parameters**

<code>winHandle</code>	Handle of window to disable.
------------------------	------------------------------

**Result** Returns nothing.

**Comments** Disabled windows ignore all pen input and cannot be made the current window or the draw window. Windows are usually disabled when they are removed from the screen.

This function does not affect the visual appearance of the window.

**See Also** [WinEnableWindow](#), [WinDeleteWindow](#)

## WinDisplayToWindowPt

**Purpose** Convert a display-relative coordinate to a window-relative coordinate. The coordinate returned is relative to the display window.

**Prototype** `void WinDisplayToWindowPt ( SWordPtr extentX,  
SWordPtr extentY)`

**Parameters** `extentX, extentY` Pointer to x and y coordinate to convert.

**Result** Returns nothing.

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

## WinDrawBitmap

**Purpose** Draw a bitmap at the given x and y coordinates.

**Prototype** `void WinDrawBitmap ( BitmapPtr bitmapP,  
SWord x,  
Sword y)`

**Parameters** `bitmapP` Pointer to a bitmap.  
`x, y` The x and y coordinates of the top-left corner.

**Result** Returns nothing.

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

## WinDrawChars

**Purpose** Draw the specified characters in the draw window.

**Prototype**

```
void WinDrawChars ( CharPtr chars,  
                    Word len,  
                    SWord x,  
                    SWord y)
```

**Parameters**

chars	Pointer to the characters to draw.
len	Number of characters to draw.
x, y	Left and top bound of first character to draw.

**Result** Returns nothing.

**Comment** Before calling this function, consider calling [WinSetUnderlineMode](#) and [FntSetFont](#).

**See Also** [WinDrawInvertedChars](#), [WinEraseChars](#), [WinSetUnderlineMode](#)

## WinDrawGrayLine

**Purpose** Draw a line in the draw window.

**Prototype**

```
void WinDrawGrayLine ( SWord x1, SWord y1,  
                      SWord x2, SWord y2)
```

**Parameters**

x1, y1	x and y coordinates of the start of the line.
x2, y2	x and y coordinate of the end of the line.

**Result** Returns nothing.

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

## WinDrawGrayRectangleFrame

- Purpose** Draw a gray rectangular frame in the draw window.
- Prototype** `void WinDrawGrayRectangleFrame ( FrameType frame,  
RectanglePtr r)`
- Parameters**
- |                    |                                    |
|--------------------|------------------------------------|
| <code>frame</code> | Type of frame to draw.             |
| <code>r</code>     | Pointer to the rectangle to frame. |
- Result** Returns nothing.
- Comments** 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](#)

## WinDrawInvertedChars

- Purpose** Draw the specified characters inverted (background color) in the draw window.
- Prototype** `void WinDrawInvertedChars ( CharPtr chars,  
Word len,  
SWord x,  
SWord y)`
- Parameters**
- |                    |  |
|--------------------|--|
| <code>chars</code> | Pointer to the characters to draw.             |
| <code>len</code>   | Number of characters to draw.                  |
| <code>x, y</code>  | Left and top bound of first character to draw. |
- Result** Returns nothing.
- Comments** The characters are drawn in the background color and the off pixels are drawn in the foreground color.  
Before calling this function, consider calling [WinSetUnderlineMode](#) and [FntSetFont](#).
- See Also** [WinDrawChars](#)

## **WinDrawLine**

**Purpose** Draw a line in the draw window.

**Prototype** `void WinDrawLine ( short x1, short y1,  
short x2, short y2)`

**Parameters** `x1, y1` x and y coordinates of the start of the line.  
`x2, y2` x and y coordinate of the end of the line.

**Result** Returns nothing.

**See Also** [WinDrawGrayLine](#), [WinEraseLine](#), [WinFillLine](#)

## **WinDrawRectangle**

**Purpose** Draw a black rectangle in the draw window; the rectangle can have square or round corners.

**Prototype** `void WinDrawRectangle ( RectanglePtr r,  
Word cornerDiam)`

**Parameters** `r` Pointer to the rectangle to draw.  
`cornerDiam` Diameter of rounded corners. Zero for square corners.

**Result** Returns nothing.

**Comments** The `cornerDiam` parameter specifies the diameter 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** [WinFillRectangle](#), [WinEraseRectangle](#)

## WinDrawRectangleFrame

- Purpose** Draw a rectangular frame around the specified region in the draw window.
- Prototype** `void WinDrawRectangleFrame ( FrameType frame,  
RectanglePtr r )`
- Parameters**
- |                    |                                    |
|--------------------|------------------------------------|
| <code>frame</code> | Type of frame to draw.             |
| <code>r</code>     | Pointer to the rectangle to frame. |
- Result** Returns nothing.
- Comments** The frame is drawn outside the specified region.
- See Also** [WinEraseRectangleFrame](#), [WinGetFramesRectangle](#), [WinDrawGrayRectangleFrame](#), [WinDrawWindowFrame](#)

## WinDrawWindowFrame

- Purpose** Draw the frame of the current drawing window.
- Prototype** `void WinDrawWindowFrame (void)`
- Parameters** None.
- Result** Returns nothing.
- See Also** [WinDrawRectangleFrame](#), [WinGetDrawWindow](#)

## **WinEnableWindow**

**Purpose** Enable a window.

**Prototype** `void WinEnableWindow (WinHandle winHandle)`

**Parameters** `winHandle` Handle of the window to enable.

**Result** Returns nothing.

**Comments** An enabled window accepts pen input and can be made the active window.

This routine does not affect the visual appearance of the window.

**See Also** [WinDisableWindow](#), [WinSetActiveWindow](#)

## **WinEraseChars**

**Purpose** Erase specified characters in the draw window.

**Prototype** `void WinEraseChars ( CharPtr chars, Word len, SWord x, SWord y)`

**Parameters** `chars` Pointer to the characters to erase.  
`len` Number of characters to erase.  
`x, y` Left and top bound of first character to erase.

**Result** Returns nothing.

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

## WinEraseLine

- Purpose** Erase a line in the draw window.
- Prototype** `void WinEraseLine ( SWord x1, SWord y1,  
SWord x2, SWord y2)`
- Parameters** `x1, y1` x and y coordinate of the start of the line.  
`x2, y2` x and y coordinate of the end of the line.
- Result** Returns nothing.
- See Also** [WinDrawLine](#)

## WinEraseRectangle

- Purpose** Erase a rectangle in the draw window. (The rectangle can have round or square corners; see [WinDrawRectangle](#).)
- Prototype** `void WinEraseRectangle ( RectanglePtr r,  
Word cornerDiam)`
- Parameters** `r` Pointer to the rectangle to erase.  
`cornerDiam` Diameter of rounded corners; zero for square corners.
- Result** Returns nothing.
- See Also** [WinDrawRectangle](#)

## **WinEraseRectangleFrame**

**Purpose** Erase a rectangular frame in the draw window.

**Prototype** `void WinEraseRectangleFrame ( FrameType frame,  
RectanglePtr r)`

**Parameters** `frame` Type of frame to erase.  
`r` Pointer to the rectangular frame.

**Result** Returns nothing.

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

## **WinEraseWindow**

**Purpose** Erase the contents of the draw window. This routine doesn't erase the frame around the draw window.

**Prototype** `void WinEraseWindow (void)`

**Parameters** None.

**Result** Returns nothing.

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

## WinFillLine

- Purpose** Fill a line in the draw window with the current pattern. You can set the current pattern with [WinSetPattern](#).
- Prototype**

```
void WinFillLine ( SWord x1, SWord y1,  
                  SWord x2, SWord y2)
```
- Parameters**

x1, y1	x and y coordinate of the start of the line.
x2, y2	x and y coordinate of the end of the line.
- Result** Returns nothing.
- See Also** [WinSetPattern](#), [WinDrawLine](#)

## WinFillRectangle

- Purpose** Draw a rectangle with current pattern. (The rectangle can have square or round corners.)
- Prototype**

```
void WinFillRectangle ( RectanglePtr r,  
                       Word cornerDiam)
```
- Parameters**

r	Pointer to the rectangle to draw.
cornerDiam	Diameter of rounded corners. Zero for square corners.
- Result** Returns nothing.
- Comments** You can set the current pattern with [WinSetPattern](#).
- See Also** [WinSetPattern](#), [WinDrawRectangle](#)

## **WinGetActiveWindow**

**Purpose** Return the window handle of the active window.

**Prototype** `WinHandle WinGetActiveWindow (void)`

**Parameters** None.

**Result** Returns the handle of the active window.

**See Also** [WinSetActiveWindow](#), [WinGetDisplayWindow](#),  
[WinGetFirstWindow](#), [WinGetDrawWindow](#), [WinRemoveWindow](#)

## **WinGetClip**

**Purpose** Return the clipping rectangle of the draw window.

**Prototype** `void WinGetClip (RectanglePtr r)`

**Parameters** `r` Pointer to a structure to hold the clipping bounds.

**Result** Returns nothing.

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

## WinGetDisplayExtent

- Purpose** Return the width and height of the display (the screen).
- Prototype** `void WinGetDisplayExtent ( SWordPtr extentX,  
SWordPtr extentY)`
- Parameters**
- |                      |                                       |
|----------------------|---------------------------------------|
| <code>extentX</code> | Pointer to the width of the display.  |
| <code>extentY</code> | Pointer to the height of the display. |
- Result** Returns nothing.

## WinGetDisplayWindow

- Purpose** Return the window handle of the display window.
- 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 physical display (screen).
- See Also** [WinGetDisplayExtent](#), [WinGetActiveWindow](#),  
[WinGetDrawWindow](#)

## **WinGetDrawWindow**

**Purpose** Return the window handle of the current draw window.

**Prototype** WinHandle WinGetDrawWindow (void)

**Parameters** None.

**Result** Returns handle of draw window.

**See Also** [WinGetDisplayWindow](#), [WinGetActiveWindow](#),  
[WinSetDrawWindow](#)

## **WinGetFirstWindow**

**Purpose** Return a pointer to the first window in the linked list of windows.

**Prototype** WinHandle WinGetFirstWindow (void)

**Parameters** None.

**Result** Returns handle of first window.

**Comments** This function is usually used by the system only.

**See also** [WinAddWindow](#), [WinGetActiveWindow](#)

## WinGetFramesRectangle

**Purpose** Return the region needed to draw a rectangle with the specified frame around it.

**Prototype** `void WinGetFramesRectangle (FrameType frame,  
RectanglePtr r,  
RectanglePtr obscuredRect)`

**Parameters**

<code>frame</code>	Type of frame drawn around the rectangle.
<code>r</code>	Pointer to the rectangle to frame.
<code>obscuredRect</code>	Pointer to the rectangle obscured by the frame.

**Result** Returns nothing.

**Comments** Frames are always drawn around (outside) a rectangle.

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

## **WinGetPattern**

**Purpose** Return the current fill pattern.

**Prototype** `void WinGetPattern (CustomPatternType pattern)`

**Parameters** `pattern` Pattern buffer to hold pattern.

**Result** Returns nothing.

**Comments** The fill pattern is used by [WinFillLine](#) and [WinFillRectangle](#). A pattern defines an 8-x-8 bit pattern. The pattern is tiled to fill the specified region. The pattern structure is eight bytes long, the first byte is the first row of the pattern.

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

## **WinGetWindowBounds**

**Purpose** Return the bounds of the current draw window in display-relative coordinates.

**Prototype** `void WinGetWindowBounds (RectanglePtr r)`

**Parameters** `r` Pointer to a rectangle.

**Result** Returns nothing.

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

## WinGetWindowExtent

- Purpose** Return the width and height of the current draw window.
- Prototype** `void WinGetWindowExtent ( SWordPtr extentX,  
SWordPtr extentY)`
- Parameters**
- |                      |   |
|----------------------|---|
| <code>extentX</code> | Pointer to the width of the draw window.  |
| <code>extentY</code> | Pointer to the height of the draw window. |
- Result** Returns nothing.
- See Also** [WinGetWindowBounds](#), [WinGetWindowFrameRect](#),

## WinGetWindowFrameRect

- Purpose** Return a rectangle, in display-relative coordinates, that defines the size and location of a window and its frame.
- Prototype** `void WinGetWindowFrameRect ( WinHandle winHandle,  
RectanglePtr r)`
- Parameters**
- |                        |   |
|------------------------|---|
| <code>winHandle</code> | Handle of window whose coordinates are desired. |
| <code>r</code>         | Pointer to the coordinates of the window.       |
- Result** Returns nothing.
- See Also** [WinGetWindowBounds](#)

## **WinGetWindowPointer**

**Purpose** Return a pointer to the specified window's `WindowType` structure.

**Prototype** `WinPtr WinGetWindowPointer (WinHandle winHandle)`

**Parameters** `winHandle` Handle of a window.

**Result** Returns pointer to the specified window's `WindowType` structure.

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

## **WinInitializeWindow**

**Purpose** Initialize the screen-dependent members of a `WindowType` structure and set the window's clipping bounds to the window's bounds.

**Prototype** `void WinInitializeWindow (WinHandle winHandle)`

**Parameters** `winHandle` Handle of a window.

**Result** Returns nothing.

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

## WinInvertChars

**Purpose** Invert the specified characters in the draw window.

**Prototype** `void WinInvertChars ( CharPtr chars, Word len,  
SWord x, SWord y)`

**Parameters**

<code>chars</code>	Pointer to the characters to invert.
<code>len</code>	Number of characters to invert.
<code>x, y</code>	Left and top bound of first character to invert.

**Result** Returns nothing.

**See Also** [WinDrawInvertedChars](#), [WinDrawChars](#)

## WinInvertLine

**Purpose** Invert a line in the draw window.

**Prototype** `void WinInvertLine ( SWord x1, SWord y1,  
SWord x2, SWord y2)`

**Parameters**

<code>x1, y1</code>	x and y coordinate of the start of the line.
<code>x2, y2</code>	x and y coordinate of the end of the line.

**Result** Returns nothing.

**See Also** [WinInvertRectangle](#), [WinInvertRectangleFrame](#),  
[WinDrawLine](#), [WinEraseLine](#)

## **WinInvertRectangle**

**Purpose** Invert a rectangle in the draw window. (The rectangle can have square or round corners.)

**Prototype** `void WinInvertRectangle ( RectanglePtr r,  
Word cornerDiam)`

**Parameters** `r` Pointer to the rectangle to invert.  
`cornerDiam` Diameter of rounded corners; zero for square corners.

**Result** Returns nothing.

**See Also** [WinInvertLine](#), [WinInvertRectangleFrame](#),  
[WinDrawRectangle](#)

## **WinInvertRectangleFrame**

**Purpose** Invert a rectangular frame in the draw window.

**Prototype** `void WinInvertRectangleFrame ( FrameType frame,  
RectanglePtr r)`

**Parameters** `frame` Type of frame to invert.  
`r` Pointer to the rectangular frame to invert.

**Result** Returns nothing.

**See Also** [WinInvertRectangle](#), [WinInvertLine](#),  
[WinDrawRectangleFrame](#), [WinEraseRectangleFrame](#)

## WinModal

- Purpose** Return TRUE if the specified window is modal.
- Prototype** Boolean WinModal (WinHandle winHandle)
- Parameters** winHandle      Handle of a window.
- Result** Returns TRUE if modal, otherwise FALSE.
- Comments** A window is modal if it cannot lose the focus.

## WinRemoveWindow

- Purpose** Remove the specified window from the window list.
- Prototype** void WinRemoveWindow (WinHandle winHandle)
- Parameters** winHandle      Handle of a window.
- Result** Returns nothing.
- Comments** Doesn't free the memory used by the window.
- See Also** [WinAddWindow](#), [WinDeleteWindow](#), [WinGetFirstWindow](#)

## 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.

**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.

**Prototype** `void WinRestoreBits ( WinHandle winHandle,  
SWord destX,  
SWord destY)`

**Parameters** `winHandle` Handle of window to copy and delete.  
`destX, destY` x and 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](#)

## WinSaveBits

- Purpose** Create an offscreen window and copy the specified region from the draw window to the offscreen window.
- Prototype** `WinHandle WinSaveBits ( RectanglePtr sourceP,  
WordPtr error)`
- Parameters**
- |                      |   |
|----------------------|---|
| <code>sourceP</code> | Pointer to the bounds of the region to save, relative to the display. |
| <code>error</code>   | 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.
- See Also** [WinRestoreBits](#)

## **WinScrollRectangle**

**Purpose** Scroll a rectangle in the draw window.

**Prototype** `void WinScrollRectangle (RectanglePtr r,  
DirectionType direction,  
SWord distance,  
RectanglePtr vacated)`

**Parameters**

<code>r</code>	Pointer to the rectangle to scroll.
<code>direction</code>	Direction to scroll (up, down, left, or right).
<code>distance</code>	Distance to scroll in pixels.
<code>vacated</code>	Pointer to the rectangle that needs to be re-drawn 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.
- 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.
- All user input is directed to the active window.
- See Also** [WinAddWindow](#), [WinGetActiveWindow](#), `EvtGetEvent` (documented in “Developing Palm OS Applications, Part I”)

## WinSetClip

- Purpose** Set the clipping rectangle of the draw window.
- Prototype** `void WinSetClip (RectanglePtr r)`
- Parameters** `r` Pointer to a structure holding the clipping bounds.
- Result** Returns nothing.
- See Also** [WinClipRectangle](#), [WinSetClip](#), [WinGetClip](#)

## **WinSetDrawWindow**

**Purpose** Set the draw window. (All drawing operations are relative to the draw window.)

**Prototype** `WinHandle WinSetDrawWindow (WinHandle winHandle)`

**Parameters** `winHandle` Handle of a window.

**Result** Returns the draw window.

**See Also** [WinGetDrawWindow](#), [WinSetActiveWindow](#)

## **WinSetPattern**

**Purpose** Set the current fill pattern.

**Prototype** `void WinSetPattern (CustomPatternType pattern)`

**Parameters** `pattern` Pattern to use.

**Result** Returns nothing.

**Comments** The fill pattern is used by [WinFillLine](#) and [WinFillRectangle](#).

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

## WinSetUnderlineMode

- Purpose** Set the graphic state to enable or disable the underlining of characters.
- Prototype** UnderlineModeType  
WinSetUnderlineMode (UnderlineModeType mode)
- Parameters** mode                      New underline mode type, one of noUnderline, grayUnderline, solidUnderline.
- Result** Returns the previous underline mode type.
- See Also** [WinDrawChars](#)

## WinSetWindowBounds

- Purpose** Set the bounds of the window to display relative coordinates.
- Prototype** void WinSetWindowBounds ( WinHandle winHandle,  
RectanglePtr r)
- Parameters** winHandle                  Handle for the window for which to set the bounds.  
r    Pointer to a rectangle to use for bounds.
- Result** Returns nothing.

### **WinValidateHandle**

**Purpose** Return TRUE if the specified handle references a valid window object.

**Prototype** `Boolean WinValidateHandle ( const 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.

**Comments** For debugging purposes only. Do not include this function in commercial applications.

**See Also** [FrmValidatePtr](#), [FrmRemoveObject](#)

### **WinWindowToDisplayPt**

**Purpose** Convert a window-relative coordinate to a display-relative coordinate.

**Prototype** `void WinWindowToDisplayPt ( SWordPtr extentX, SWordPtr extentY )`

**Parameters** extentX, extentY    Pointer to x and y coordinate to convert.

**Result** Returns nothing.

**Comments** The coordinate passed is assumed to be relative to the draw window.

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

# Index

---

## Numerics

68328 processor 44

## A

about box 348

AbtShowAbout 348

action codes *See* launch codes

active form 278, 279

active window 207, 403, 427

active windows list 400

alarms

    and launch codes 72

    sysAppLaunchCmdTimeChange 82

alert manager 180

alert resource 89

alerts

    confirmation 89

    custom alert 275

    error 89

    system-defined 180

    warning 89

application design and conduits 34

applications

    control flow 55

    designing UI 34

    event driven 36

    initialization 42

    Security 81

appStopEvent 192

Auto-Shift (field) 108

## B

battery life, maximizing 45

Berkeley Sockets API 41

bitmap label for button 102

bitmaps

*See Also* form bitmap resource

    drawing 405

bits behind menu bar 157

blank lines in field 238

boldFont 271

button objects 128

button resource 100

    bitmap label 102

    highlighting 128

    increment arrow 101

    label 101

buttons (silk-screened buttons) 199

## C

C library functions 39

calibrating digitizer 185

calibrating the pen 350

carriage returns 135

catalog resources 98

categories, setting label 300

CategoryCreateList 209

CategoryCreateListV10 210

CategoryEdit 211

CategoryEditV10 213

CategoryEditV20 212

CategoryFind 213

CategoryFreeList 214

CategoryFreeListV10 215

CategoryGetName 215

CategoryGetNext 216

CategoryInitialize 216

CategorySelect 217

CategorySelectV10 218

CategorySetName 219

CategorySetTriggerLabel 219

CategoryTruncateName 220

changing resources 87

character attribute functions 309–310

characters

*See Also* functions starting with Fnt

    drawing in window 406

    erasing 410

    inverting 421

    sorting text 310

check box object 128

check box resource 103

    Group ID 104

    toggle area 104

clipboard 232, 249

## Index

---

ClipboardAddItem 220  
ClipboardGetItem 221  
clipping rectangle 427  
clock, real-time 46  
code #0 resource 43  
code #1 resource 42  
CodeWarrior IDE 40  
conduits and application design 34  
Confirmation alert 89  
connectivity 45  
connector (external) 46  
Constructor 35, 86  
    catalog resources 98  
control flow 55  
control objects 128  
    and pen tracking 202  
    drawing 221  
    erasing 222  
    events 129  
    fields of structure 132  
    function list 133  
    structure 130  
ControlType structure. *See Also* control objects  
conventions for naming 43  
coordinates, display-relative vs. window-  
    relative 405  
creating active window 403  
creating modal window 403  
creating resources 86  
creator ID 77  
CtlDrawControl 221  
CtlEnabled 222  
ctlEnterEvent 192  
CtlEraseControl 222  
ctlExitEvent 192  
CtlGetLabel 223  
CtlGetValue 223  
CtlHandleEvent 129, 224  
CtlHideControl 225  
CtlHitControl 225  
CtlNewControl 226  
ctlRepeatEvent 193  
ctlSelectEvent 193  
CtlSetEnabled 228  
CtlSetLabel 228

CtlSetUsable 229  
CtlSetValue 229  
CtlShowControl 230  
CtlValidatePointer 230  
current time 82  
custom fill patterns, getting 418  
custom UI element 110

## D

data #0 resource 43  
date 367  
date system resource 134  
day selector object 194  
DayHandleEvent 348  
daySelectEvent 194  
designing UI 34  
dialogs  
    command buttons 93  
    Edit Categories 211  
    info dialog 348  
    placement 92  
digitizer 181  
    after reset 48  
    and pen manager 185  
    and PenResetCalibration function 351  
    and penUpEvent 202  
    calibrating 185  
    dimensions 47  
    polling 46  
    sampling accuracy 47  
down arrow 136  
doze mode 45  
drag-selecting and fldChangedEvent 194  
draw window 428  
drawing rectangular frame 407, 409  
drawItemsCallback 155  
drivers, restarting 47  
dynamic heap 43  
    soft reset 47  
    space requirement 42  
dynamic scrolling 205

## E

Edit Categories dialog 211  
enabled window 410

- 
- enabling windows 403
  - erasing characters 410
  - erasing lines in window 411
  - erasing rectangle 411
  - Error alert 89
  - event loop 61–64
    - example 61
  - event-driven applications 36, 55
  - events 191–207
    - naming conventions 43
    - overview 57–65
  - examples
    - event loop 61
    - startup routine 58
    - stop routine 65
  - F**
  - field objects 134
    - and text height 242
    - different keyDownEvents 136
    - dynamic resizing 195
    - events 135
    - field attributes 139
    - fields 138
    - function list 140
    - line feeds vs. carriage returns 135
    - modifying 233
    - structure 136
  - field resource 106
    - Auto-Shift 108
    - Has Scrollbar 108
  - FieldType structure 136
  - fill patterns
    - setting 428
  - fill patterns, getting 418
  - Find (global find) 74, 76
    - saving data 80
  - Find (lookup) 78
  - Find icon 199
  - flags, launch flags 67, 82
  - FldCalcFieldHeight 231
  - fldChangedEvent 194
    - in 2.0 49
  - FldCompactText 231
  - FldCopy 232
  - FldCut 232
  - FldDelete 233
  - FldDirty 233
  - FldDrawField 234
  - fldEnterEvent 195
  - FldEraseField 234
  - FldFreeMemory 235
  - FldGetAttributes 235
  - FldGetBounds 236
  - FldGetFont 236
  - FldGetInsPtPosition 237
  - FldGetMaxChars 237
  - FldGetNumberOfBlankLines 238
  - FldGetScrollPosition 238
  - FldGetScrollValues 239
  - FldGetSelection 240
  - FldGetTextAllocatedSize 241
  - FldGetTextHandle 241
  - FldGetTextHeight 242
  - FldGetTextLength 242
  - FldGetTextPtr 243
  - FldGetVisibleLines 243
  - FldGrabFocus 244
  - FldHandleEvent 135, 245
  - fldHeightChangedEvent 195
  - FldInsert 246
  - FldMakeFullyVisible 246
  - FldNewField 247
  - FldPaste 249
  - FldRecalculateField 250
  - FldReleaseFocus 250
  - FldScrollable 251
  - FldScrollField 251
  - FldSendChangeNotification 252
  - FldSendHeightChangeNotification 252
  - FldSetAttributes 253
  - FldSetBounds 253
  - FldSetDirty 254
  - FldSetFont 254
  - FldSetInsertionPoint 255
  - FldSetInsPtPosition 255
  - FldSetMaxChars 256
  - FldSetScrollPosition 256
  - FldSetSelection 257



- 
- frmLoadEvent 197
  - FrmNewBitmap 292
  - FrmNewForm 293
  - FrmNewGadget 295
  - FrmNewLabel 296
  - frmOpenEvent 197
  - FrmPointInTitle 297
  - FrmPopupForm 297
  - FrmRemoveObject 298
  - FrmReturnToForm 299
  - FrmSaveAllForms 299
  - frmSaveEvent 197
  - FrmSetActiveForm 300
  - FrmSetCategoryLabel 300
  - FrmSetControlGroupSelection 301
  - FrmSetControlValue 301
  - FrmSetEventHandler 302
  - FrmSetFocus 302
  - FrmSetGadgetData 303
  - FrmSetMenu 303
  - FrmSetNotUserModified 304
  - FrmSetObjectBounds 304
  - FrmSetObjectPosition 305
  - FrmSetTitle 306
  - FrmShowObject 306
  - frmTitleEnterEvent 198
  - frmTitleSelectEvent 198
  - frmUpdateEvent 197
  - FrmUpdateForm 307
  - FrmUpdateScrollers 307
  - FrmValidatePtr 308
  - FrmVisible 308
  - function naming conventions 43
- G**
- gadget resource 110
    - FrmSetGadgetData 303
  - GetCharAttr 309
  - GetCharCaselessValue 309
  - GetCharSortValue 310
  - global find 74, 76
    - saving data 80
  - global variables
    - and launch codes 66
    - erasing 47
  - goto (global find) 76
  - Graffiti
    - Command shortcuts 201
    - customizing behavior 181
    - Help 183
    - Help character 183
  - Graffiti manager 181
    - functions 311–323
  - Graffiti Shift
    - functions 323–325
    - getting and setting state 182
    - Indicator resource 111
  - Graffiti ShortCuts
    - database 182
    - SysShortCutListDialog 322
  - graffitiReferenceChr 183
  - grayUnderline 429
  - Gremlins 40
  - GrfAddMacro 311
  - GrfAddPoint 311
  - GrfCleanState 312
  - GrfDeleteMacro 312
  - GrfFilterPoints 313
  - GrfFindBranch 313
  - GrfFlushPoints 314
  - GrfGetAndExpandMacro 314
  - GrfGetGlyphMapping 315
  - GrfGetMacro 316
  - GrfGetMacroName 316
  - GrfGetNumPoints 317
  - GrfGetPoint 317
  - GrfGetState 318
  - GrfInitState 319
  - GrfMatch 319
  - GrfMatchGlyph 320
  - GrfProcessStroke 182, 321
  - GrfSetState 322
  - Group ID 104
  - groups of controls 279
  - GsiEnable 323
  - GsiEnabled 324
  - GsiInitialize 324

## Index

---

GsiSetLocation 324

GsiSetShiftState 325

GUI *See* UI

## H

hard reset 47, 48, 81

hardware button presses and key manager 184

hardware overview 44

Has Scrollbar (field) 108

height of text in field 242

Help ID 90

highlighting button objects 128

HotSync and sysAppLaunchCmdSyncNotify 80

## I

icons

    alert 89

    Find icon 199

IDE 40

increment arrow 101

Information alert 89

initialization 78

    global variables 58

    of application 42

insertion point functions 326–328

insertion point object 151

insertion points

    and FldGetInsPtPosition 237

    and FldGrabFocus 244

    and FldReleaseFocus 250

    and FldSetInsertionPoint 255

    displayed in field 234

InsPtEnable 326

InsPtEnabled 326

InsPtGetHeight 327

InsPtGetLocation 327

InsPtSetHeight 328

InsPtSetLocation 328

inverting characters in draw window 421

inverting line in draw window 421

inverting rectangles 422

## K

key manager 184

key manager functions 329–330

KeyCurrentState 184, 329

keyDownEvent 184, 199

    in field object 136

KeyRates 184, 330

KeySetMask 349

## L

label (button) 101

label resource 112

    bitmap label for button 102

    wrapping text 112

largeBoldFont 271

launch codes 36, 66–84

    and global variables 66

    code example 68

    creating 84

    launch flags 67

    parameter blocks 67

    predefined 84

    responding 83

    summary 70

    use by application 66

    use by system 66

launch flags 67, 82

Layout Appearance panel 87

Layout Properties panel 87

LCD screen 46

left arrow 136

line feeds 135

lines

    erasing 411

    inverting 421

list objects

    and pen tracking 202

    creating category list 209

    drawItemsCallback 155

    events 153

    fields 155

    function list 156

    functions 332–341

    scroll indicators 152

    structure 154

- 
- list resource
    - and popup trigger 114
    - event flow 114
    - vs. menu resource 114
  - ListDrawDataFuncType 155
  - ListType structure 154
  - LocGetNumberSeparators 72, 349
  - locking system 81
  - lookup 78
    - example 79
  - LstDrawList 332
  - lstEnterEvent 200
  - LstEraseList 332
  - lstExitEvent 200
  - LstGetNumberOfItems 333
  - LstGetSelection 333
  - LstGetSelectionText 334
  - LstGetVisibleItems 334
  - LstHandleEvent 335
  - LstMakeItemVisible 336
  - LstNewList 336
  - LstPopupList 338
  - LstScrollList 338
  - lstSelectEvent 200
  - LstSetDrawFunction 339
  - LstSetHeight 339
  - LstSetListChoices 340
  - LstSetPosition 340
  - LstSetSelection 341
  - LstSetTopItem 341
- M**
- Makefile 41
  - managers
    - overview 37
    - vs. library 179
  - mapping, resources and UI objects 36
  - maximizing battery life 45
  - memory
    - and FldCompactText 231
    - and FldFreeMemory 235
    - and FldSetText 258
  - menu bar objects 157
  - menu bars 94
    - and user actions 157
    - bits behind 157
  - Menu Item Object fields 161
  - menu objects
    - See Also* menus 158
    - events 158
    - fields 160
    - function list 162
    - structure 158
  - menu pulldown object 161
  - MenuDispose 342
  - MenuDrawMenu 343
  - MenuEraseStatus 344
  - menuEvent 201
  - MenuGetActiveMenu 345
  - MenuHandleEvent 346
  - MenuInit 347
  - menus 94
    - active area 97
    - adding to form 96
    - creating 95
    - event flow 98
    - FrmSetMenu 303
    - functions 342–347
    - shortcut key 96
    - user interaction 96
  - MenuSetActiveMenu 347
  - Missing Character Symbol 265
  - modal form 91
  - modal window 338, 403, 423
  - modes 44
  - modified field objects 233
  - modifying Graffiti shortcuts 183
  - Motorola 68328 44
- N**
- naming conventions 43
  - nilEvent 201
  - noUnderline 429
- O**
- Object Identifier 100
  - off-screen windows 402

## Index

---

### P

- Palm OS 2.0 48
- Palm OS 3.0 48
- PalmPilot Professional memory 44
- parameter blocks 67
- patches, loading during reset 48
- pen location polling 47
- pen manager 185
- pen manager functions 350–351
- pen queue 185
- PenCalibrate 350
- penDownEvent 202
- penMoveEvent 202
- PenResetCalibration 351
- penUpEvent 202
- PilotMain 66
- popSelectEvent 203
- popup list 338
- popup trigger
  - event flow 116
- popup trigger object 128
- popup trigger resource 115
  - and list 114
- predefined launch codes 84
- pref #0 resource 42
- preferences
  - application-specific 58
  - changing with launch codes 79
  - short cuts 183
  - system 58
- PrgHandleEvent 353
- PrgStartDialog 354
- PrgStopDialog 355
- PrgUpdateDialog 356
- PrgUserCancel 357
- processor 44
- progress manager 186
- push button object 128
- push button resource 117
  - creating row 119
  - event flow 119

### R

- radio button *See* push button 119
- RAM 44
- RctCopyRectangle 358
- RctGetIntersection 358
- RctInsetRectangle 359
- RctOffsetRectangle 360
- RctPtInRectangle 361
- RctSetRectangle 361
- real-time clock 46
- rectangles
  - copying 358
  - erasing 411
  - intersecting 358
  - inverting 422
  - moving 360
  - resizing 359
  - scrolling 426
- RectangleType structure 128
- repeat control object 128
  - and ctlRepeatEvent 193
- repeating button resource 120
  - event flow 121
- ResEdit 86
  - resource naming conventions 43
- reset 47, 81
  - digitizer screen 48
  - hard reset 48
  - loading patches 48
  - soft reset 47
- resources
  - See Also* UI resources
  - alert 89
  - and UI objects 36
  - changing 87
  - check box 103
  - field 106
  - form 91
  - form bitmap 109
  - gadget 110
  - Graffiti Shift Indicator 111
  - label 112
  - menu 94
  - menu bar 94
  - popup trigger 115
  - push button 117

- 
- repeating button 120
  - scrollbar 122
  - selector trigger 124
  - string 94
  - table 126
- retrieving system version number 51
- right arrow 136
- ROM 44
- rounded corners 411, 413
- running mode 45
- ## S
- Save Behind 91
- ScI DrawScrollBar 362
- sclEnterEvent 204
- sclExitEvent 204
- ScI GetScrollBar 362
- ScI HandleEvent 363
- sclRepeatEvent 205
  - and sclExitEvent 204
- ScI SetScrollBar 364
- ScrDisplayMode 396
- screen command buttons 93
- screen layout design 34
- screen size 46
- scroll arrows
  - FrmUpdateScrollers 307
- scroll position in field 238
- scrollbar functions 362–364
- scrollbar objects 162
  - fields 164
  - in tables 378
  - structure 163
- scrollbar resource 122
- scrolling rectangle in window 426
- Security application 81
- SelectDay 367
- SelectDayV10 367
- selection in field 240
- selector trigger object 128
- selector trigger resource 124
- SelectTime 368
- serial communication 45
- shortcut key 96
- shortcuts, Graffiti 182
- sleep mode 44
  - and current time 46
- soft reset 47, 81
  - dynamic heap 47
- solidUnderline 429
- sorting text 310
- square corners 411, 413
- stack size requirement 42
- startup routine, example 58
- stdFont 271
- stop routine example 65
- storage heaps, erasing 48
- StrDelocalizeNumber, and launch code 72
- string resource 94
- StrLocalizeNumber, and launch code 72
- structure elements, naming convention 43
- structure of field object 136
- summary of launch codes 70
- sysAppLaunchCmdAlarmTriggered 72
- sysAppLaunchCmdCountryChange 72
- sysAppLaunchCmdDisplayAlarm 72
- sysAppLaunchCmdExgAskUser 72
- sysAppLaunchCmdExgReceiveData 74
- sysAppLaunchCmdFind 74
- sysAppLaunchCmdGoto 76
- sysAppLaunchCmdInitDatabase 77
- sysAppLaunchCmdLookup 78
- sysAppLaunchCmdPanelCalledFromApp 79
- SysAppLaunchCmdReset 47
- sysAppLaunchCmdReturnFromPanel 80
- sysAppLaunchCmdSaveData 80
- sysAppLaunchCmdSyncNotify 80
- sysAppLaunchCmdSystemLock 81
- sysAppLaunchCmdSystemReset 81
- sysAppLaunchCmdTimeChange 82
- sysAppLaunchFlagNewGlobals launch flag 82
- sysAppLaunchFlagNewStack launch flag 82
- sysAppLaunchFlagNewThread launch flag 82
- sysAppLaunchFlagSubCal launch flag 83
- sysAppLaunchFlagUIApp launch flag 83
- SysGraffitiReferenceDialog 183
- SysGrfShortCutListDialog 322
- system preferences 58

## Index

---

system tick interrupts 46  
system version number 51

### T

table functions 369–395  
table objects 166  
    fields 168  
    structure 166  
table resource 126  
    maximum size 126  
tables  
    setting load data callback 391  
    setting save data callback 395  
TblDrawTable 369  
TblEditing 369  
tblEnterEvent 205  
TblEraseTable 370  
tblExitEvent 206  
TblFindRowData 370  
TblFindRowID 371  
TblGetBounds 371  
TblGetColumnSpacing 372  
TblGetColumnWidth 372  
TblGetCurrentField 373  
TblGetItemBounds 373  
TblGetItemFont 374  
TblGetItemInt 374  
TblGetLastUsableRow 375  
TblGetNumberOfRows 375  
TblGetRowData 375  
TblGetRowHeight 376  
TblGetRowID 376  
TblGetSelection 377  
TblGrabFocus 377  
TblHandleEvent 378  
TblHasScrollBar 378  
TblInsertRow 379  
TblMarkRowInvalid 379  
TblMarkTableInvalid 380  
TblRedrawTable 380  
TblReleaseFocus 381  
TblRemoveRow 381  
TblRowInvalid 382  
TblRowSelectable 382  
TblRowUsable 383  
tblSelectEvent 206  
TblSelectItem 383  
TblSetBounds 384  
TblSetColumnEditIndicator 384  
TblSetColumnSpacing 385  
TblSetColumnUsable 385  
TblSetColumnWidth 386  
TblSetCustomDrawProcedure 387  
TblSetItemFont 388  
TblSetItemInt 388  
TblSetItemPtr 389  
TblSetItemStyle 390  
TblSetLoadDataProcedure 391  
TblSetRowData 392  
TblSetRowHeight 392  
TblSetRowID 393  
TblSetRowSelectable 393  
TblSetRowStaticHeight 394  
TblSetRowUsable 394  
TblSetSaveDataProcedure 395  
TblUnhighlightSelection 395  
text clipboard 232  
text, finding with GetCharCaselessValue 309  
time system resource 134  
time, displaying and selecting 368  
timer 46  
title (form) 93  
titles  
    active area 198  
    copying form title 274

### U

UI design 34  
UI objects 127–175  
    buttons 128  
    check box 128  
    control objects 128  
    field 134  
    form 142  
    insertion point 151  
    list 152  
    menu bars 157  
    popup trigger 128  
    push button 128

---

- selector trigger 128
- table 166
- windows 171

UI resources 86–126

- creating 86
- custom 110

up arrow 136

user interaction design 34

## V

version number (system) 51

## W

Warning alert 89

WinAddWindow 400

WinClipRectangle 400

WinCopyRectangle 401

WinCreateOffscreenWindow 402

WinCreateWindow 403

WinDeleteWindow 404

WinDisableWindow 404

WinDisplayToWindowPt 405

window list 280

window objects 171

- fields of structure 172
- function list 173
- off-screen 171
- structure 171

windows

- active window 207
- enabled window 410

WinDrawBitmap 405

WinDrawChars 406

WinDrawGrayLine 406

WinDrawGrayRectangleFrame 407

WinDrawInvertedChars 407

WinDrawLine 408

WinDrawRectangle 408

WinDrawRectangleFrame 409

WinDrawWindowFrame 409

WinEnableWindow 410

winEnterEvent 207

WinEraseChars 410

WinEraseLine 411

WinEraseRectangle 411

WinEraseRectangleFrame 412

WinEraseWindow 412

winExitEvent 207

WinFillLine 413

WinFillRectangle 413

WinGetActiveWindow 414

WinGetClip 414

WinGetDisplayExtent 415

WinGetDisplayWindow 415

WinGetDrawWindow 416

WinGetFirstWindow 416

WinGetFramesRectangle 417

WinGetPattern 418

WinGetWindowBounds 418

WinGetWindowExtent 419

WinGetWindowFrameRect 419

WinGetWindowPointer 420

WinInitializeWindow 420

WinInvertChars 421

WinInvertLine 421

WinInvertRectangle 422

WinInvertRectangleFrame 422

WinModal 423

WinRemoveWindow 423

WinResetClip 424

WinRestoreBits 424

WinSaveBits 425

WinScrollRectangle 426

WinSetActiveWindow 427

WinSetClip 427

WinSetDrawWindow 428

WinSetPattern 428

WinSetUnderlineMode 429

WinSetWindowBounds 429

WinType structure 171

WinValidateHandle 430

WinWindowToDisplayPt 430

word wrap 270

## Index

---