



Adobe Dialog Manager Programmer's Guide and Reference



Adobe Dialog Manager

ADOBE SYSTEMS INCORPORATED

Corporate Headquarters


345 Park Avenue

San Jose, CA 95110-2704

(408) 536-6000

<http://partners.adobe.com>

September, 2002



Copyright 2002 Adobe Systems Incorporated. All rights reserved.

NOTICE: All information contained herein is the property of Adobe Systems Incorporated. No part of this publication (whether in hardcopy or electronic form) may be reproduced or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of the Adobe Systems Incorporated.

PostScript is a registered trademark of Adobe Systems Incorporated. All instances of the name PostScript in the text are references to the PostScript language as defined by Adobe Systems Incorporated unless otherwise stated. The name PostScript also is used as a product trademark for Adobe Systems' implementation of the PostScript language interpreter.

Except as otherwise stated, any reference to a "PostScript printing device," "PostScript display device," or similar item refers to a printing device, display device or item (respectively) that contains PostScript technology created or licensed by Adobe Systems Incorporated and not to devices or items that purport to be merely compatible with the PostScript language.

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Acrobat Capture, Acrobat Catalog, Acrobat Exchange, Acrobat Reader, Acrobat Search, Distiller, PostScript, and the PostScript logo are trademarks of Adobe Systems Incorporated.

Apple, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. PowerPC is a registered trademark of IBM Corporation in the United States. ActiveX, Microsoft, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries. UNIX is a registered trademark of The Open Group. All other trademarks are the property of their respective owners.

This publication and the information herein is furnished AS IS, is subject to change without notice, and should not be construed as a commitment by Adobe Systems Incorporated. Adobe Systems Incorporated assumes no responsibility or liability for any errors or inaccuracies, makes no warranty of any kind (express, implied, or statutory) with respect to this publication, and expressly disclaims any and all warranties of merchantability, fitness for particular purposes, and noninfringement of third party rights.



Contents

Preface23
Introduction.	23
Conventions	23
Accessing Suites.	24
Supporting Documents.	24
 Chapter 1 ADM Overview25
About Adobe Dialog Manager	25
ADM Design	25
PICA Plug-ins	25
ADM Objects in General.	27
A Quick Summary of Using ADM	29
Types.	30
Events	33
Properties	33
Resources	35
The Suites	36
ADM Object Specifics	37
ADM Dialog Objects	37
Dialog Item Objects	40
Composite Items.	52
ADM Item Groups	52
ADM Item Numeric Properties.	53
ADM Lists and Entries.	53
ADM Hierarchy Lists and List Entries	55
Macintosh and Windows ADM Item Resource Lists	56
Event Callbacks	63
Using Event Callbacks.	63
Using Init Functions	64
Using Drawer Functions	65
Using Notifier Functions	66
Using Tracker Functions	67
Using Destroy Functions.	68
Using Resizable Windows	68

Custom Item Types.	69
Using Custom Items	70
Using Timer Procedures	71
Using the C++ Interfaces.	71
Getting Started With ADM Plug-In Development	72
The General Development Process	73
Chapter 2 Using ADM with Adobe Acrobat	75
Introduction.	75
Using ADM with Acrobat	75
Initializing ADM	76
Acquiring and Using ADM Suites	76
Using Modal Dialogs.	78
Handling Resources	79
Macintosh Issues.	80
Carbon	80
'carb' Resources.	80
Development Environment and Documentation	80
Chapter 3 Using ADM with Adobe Photoshop	81
Introduction.	81
Frame Select Photoshop Plug-in.	81
Platform-Specific Resources.	81
Acquiring the Suites	87
Building, Presenting, and Using the Dialog	88
Operation	89
Chapter 4 Using ADM with Adobe Illustrator	99
A Modeless Dialog Example Using Illustrator	99
Chapter 5 Using ADM with Adobe After Effects	113
Introduction.	113
Implementation notes	113
Easy_Cheese Plug-in	113
Chapter 6 The ADM Basic Suite	115

About the ADM Basic Suite115
Accessing the Suite115
ADM Basic Suite Functions116
Resource Functions116
sADMBasic->GetIndexString()116
sADMBasic->SetPlatformCursor().117
User Interface Functions117
sADMBasic->AboutBox()117
sADMBasic->AreToolTipsEnabled().118
sADMBasic->AreToolTipsSticky().119
sADMBasic->Beep().119
sADMBasic->ChooseColor()119
sADMBasic->EnableToolTips()120
sADMBasic->ErrorAlert()121
sADMBasic->GetToolTipDelays().121
sADMBasic->HideToolTip().122
sADMBasic->LightweightErrorAlert().122
sADMBasic->MessageAlert().123
sADMBasic->PluginAboutBox().123
sADMBasic->QuestionAlert().124
sADMBasic->SetAlertButtonText().125
sADMBasic->ShowToolTip().125
sADMBasic->StandardGetFileDialog().126
sADMBasic->StandardGetDirectoryDialog().128
sADMBasic->StandardPutFileDialog().129
sADMBasic->YesNoAlert().130
Utility Functions131
sADMBasic->ADMColorToRGBColor().131
sADMBasic->GetAppFPS().131
sADMBasic->GetAppUnits().132
sADMBasic->GetDefaultIncrements().132
sADMBasic->GetLastADMErrors().133
sADMBasic->GetNumbersArePoints().133
sADMBasic->GetPaletteLayoutBounds().134
sADMBasic->GetScreenDimensions().134
sADMBasic->GetWorkspaceBounds().135
sADMBasic->LookUpZString().135
sADMBasic->SetAppFPS().136
sADMBasic->SetAppUnits().136
sADMBasic->SetDefaultIncrements().137
sADMBasic->SetNumbersArePoints().138
sADMBasic->StringToValue().138
sADMBasic->ValueToString().139
Contextual Menu Functions139
sADMBasic->CreateMenu().140

sADMBasic->DestroyMenu()140
sADMBasic->DisplayMenu()141

Chapter 7 The ADM Dialog Suite 143

About the ADM Dialog Suite143
Accessing the Suite143
Dialog Basics: Styles143
Dialog Basics: Standard Dialog Item IDs144
Dialog Basics: Callbacks.145
ADM Dialog Suite Functions145
sADMDialog->AbortTimer()145
sADMDialog->Activate()146
sADMDialog->AdjustItemTabOrder()146
sADMDialog->Create()147
sADMDialog->CreateGroupInDialog().150
sADMDialog->CreateItem()151
sADMDialog->CreateTimer()152
sADMDialog->DefaultDraw()153
sADMDialog->DefaultNotify()154
sADMDialog->DefaultTrack()154
sADMDialog->Destroy()155
sADMDialog->DestroyItem()156
sADMDialog->DisplayAsModal()156
sADMDialog->DisplayAsPopupModal()157
sADMDialog->Enable()158
sADMDialog->EndModal().158
sADMDialog->GetBoundsRect()159
sADMDialog->GetCancelItemID()159
sADMDialog->GetCursorID()160
sADMDialog->GetDefaultItemID().160
sADMDialog->GetDestroyProc()161
sADMDialog->GetDialogName()161
sADMDialog->GetDialogStyle()162
sADMDialog->GetDrawProc().162
sADMDialog->GetFont().163
sADMDialog->GetHorizontalIncrement()163
sADMDialog->GetID()164
sADMDialog->GetItem().165
sADMDialog->GetLocalRect().165
sADMDialog->GetMask()166
sADMDialog->GetMaxHeight()166
sADMDialog->GetMaxWidth().167
sADMDialog->GetMinHeight().167
sADMDialog->GetMinWidth()168

sADMDialog->GetNextItem()	168
sADMDialog->GetNotifierData()	169
sADMDialog->GetNotifyProc()	169
sADMDialog->GetPluginRef()	170
sADMDialog->GetPreviousItem()	170
sADMDialog->GetText()	171
sADMDialog->GetTextLength()	172
sADMDialog->GetTrackProc()	172
sADMDialog->GetUserData()	173
sADMDialog->GetVerticalIncrement()	173
sADMDialog->GetWindowRef()	174
sADMDialog->Invalidate()	174
sADMDialog->InvalidateRect()	175
sADMDialog->IsActive()	175
sADMDialog->IsCollapsed()	176
sADMDialog->IsDialogContextHidden()	176
sADMDialog->IsEnabled()	177
sADMDialog->IsForcedOnScreen()	177
sADMDialog->IsUpdateEnabled()	178
sADMDialog->IsVisible()	178
sADMDialog->LoadToolTips()	179
sADMDialog->LocalToScreenPoint()	179
sADMDialog->LocalToScreenRect()	180
sADMDialog->Modal()	180
sADMDialog->Move()	182
sADMDialog->PopupModal()	182
sADMDialog->RegisterItemType()	183
sADMDialog->ScreenToLocalPoint()	184
sADMDialog->ScreenToLocalRect()	184
sADMDialog->SendNotify()	185
sADMDialog->SetBoundsRect()	185
sADMDialog->SetCancelItemID()	186
sADMDialog->SetCursorID()	187
sADMDialog->SetDefaultItemID()	188
sADMDialog->SetDestroyProc()	188
sADMDialog->SetDialogName()	189
sADMDialog->SetDialogStyle()	189
sADMDialog->SetDrawProc()	190
sADMDialog->SetFont()	191
sADMDialog->SetForcedOnScreen()	191
sADMDialog->SetHorizontalIncrement()	192
sADMDialog->SetLocalRect()	192
sADMDialog->SetMask()	193
sADMDialog->SetMaxHeight()	193
sADMDialog->SetMaxWidth()	194
sADMDialog->SetMinHeight()	195
sADMDialog->SetMinWidth()	195

sADMDialog->SetNotifierData()196
sADMDialog->SetNotifyProc()197
sADMDialog->SetText()197
sADMDialog->SetTrackProc()198
sADMDialog->SetUpdateEnabled()199
sADMDialog->SetUserData()199
sADMDialog->SetVerticalIncrement()200
sADMDialog->SetWindowRef()200
sADMDialog->Show()201
sADMDialog->Size()202
sADMDialog->UnregisterItemType()202
sADMDialog->Update()203
ADM Help Support203
sADMDialog->GetHelpID()203
sADMDialog->Help()204
sADMDialog->SetHelpID()204
Chapter 8 The ADM Dialog Group Suite	207
About the ADM Dialog Group Suite207
Accessing the Suite207
The ADM Dialog Group Suite's Position Code and Group Name207
ADM DialogGroup Suite Functions208
sADMDialogGroup->CountDialogs()208
sADMDialogGroup->GetDialogGroupInfo()208
sADMDialogGroup->GetDialogName()209
sADMDialogGroup->GetNamedDialog()209
sADMDialogGroup->GetNthDialog()210
sADMDialogGroup->IsCollapsed()210
sADMDialogGroup->IsDockVisible()211
sADMDialogGroup->IsFrontTab()211
sADMDialogGroup->IsStandAlonePalette()212
sADMDialogGroup->SetDialogGroupInfo()212
sADMDialogGroup->SetTabGroup()213
sADMDialogGroup->ShowAllFloatingDialogs()214
sADMDialogGroup->ToggleAllButNoCloseFloatingDialogs()214
sADMDialogGroup->ToggleAllFloatingDialogs()214
Chapter 9 The ADM Drawer Suite	217
About the ADM Drawer Suite217
Accessing the Suite217
ADM Drawer Functions218
Using ADM Drawer Functions218

Fonts and Colors.219
The Drawer Coordinate Space219
Drawing Modes220
ADM Drawer Suite Functions221
sADMDrawer->Clear().221
sADMDrawer->ClearRect().221
sADMDrawer->Create().221
sADMDrawer->Destroy().222
sADMDrawer->DrawADMImage().223
sADMDrawer->DrawADMImageCentered().223
sADMDrawer->DrawAGMImage().224
sADMDrawer->DrawDownArrow().225
sADMDrawer->DrawIcon().225
sADMDrawer->DrawIconCentered().226
sADMDrawer->DrawLeftArrow().227
sADMDrawer->DrawLine().227
sADMDrawer->DrawOval().228
sADMDrawer->DrawPolygon().228
sADMDrawer->DrawRaisedRect().229
sADMDrawer->DrawRecoloredIcon().230
sADMDrawer->DrawRecoloredIconCentered().231
sADMDrawer->DrawRecoloredResPicture().231
sADMDrawer->DrawRecoloredResPictureCentered().232
sADMDrawer->DrawRect().233
sADMDrawer->DrawResPicture().234
sADMDrawer->DrawResPictureCentered().235
sADMDrawer->DrawRightArrow().235
sADMDrawer->DrawSunkenRect().236
sADMDrawer->DrawText().237
sADMDrawer->DrawTextCentered().237
sADMDrawer->DrawTextInaBox().238
sADMDrawer->DrawTextLeft().239
sADMDrawer->DrawTextRight().239
sADMDrawer->DrawUpArrow().240
sADMDrawer->FillOval().241
sADMDrawer->FillPolygon().241
sADMDrawer->FillRect().242
sADMDrawer->GetADMColor().242
sADMDrawer->GetADMWindowPort().243
sADMDrawer->GetAGMPort().244
sADMDrawer->GetBoundsRect().244
sADMDrawer->GetClipRect().245
sADMDrawer->GetDrawMode().245
sADMDrawer->GetFont().246
sADMDrawer->GetFontInfo().246
sADMDrawer->GetOrigin().247

sADMDrawer->GetPortRef()	.247
sADMDrawer->GetResPictureBounds()	.248
sADMDrawer->GetRGBColor()	.248
sADMDrawer->GetTextRectHeight()	.249
sADMDrawer->GetTextWidth()	.249
sADMDrawer->GetThisFontInfo()	.250
sADMDrawer->GetUpdateRect()	.250
sADMDrawer->IntersectClipPolygon()	.251
sADMDrawer->IntersectClipRect()	.252
sADMDrawer->InvertRect()	.252
sADMDrawer->ReleaseADMWindowPort()	.253
sADMDrawer->SetADMColor()	.253
sADMDrawer->SetClipPolygon()	.254
sADMDrawer->SetClipRect()	.254
sADMDrawer->SetDrawMode()	.255
sADMDrawer->SetFont()	.255
sADMDrawer->SetOrigin()	.256
sADMDrawer->SetRGBColor()	.256
sADMDrawer->SubtractClipPolygon()	.257
sADMDrawer->SubtractClipRect()	.258
sADMDrawer->UnionClipPolygon()	.258
sADMDrawer->UnionClipRect()	.259

Chapter 10 The ADM Entry Suite 261

About the ADM Entry Suite	.261
Accessing the Suite	.261
Initializing An Entry	.261
ADM Entry Suite Functions	.262
sADMEEntry->AbortTimer()	.262
sADMEEntry->Activate()	.262
sADMEEntry->Check()	.263
sADMEEntry->Create()	.263
sADMEEntry->CreateTimer()	.264
sADMEEntry->DefaultDraw()	.265
sADMEEntry->DefaultNotify()	.266
sADMEEntry->DefaultTrack()	.266
sADMEEntry->Destroy()	.267
sADMEEntry->Enable()	.268
sADMEEntry->GetBoundsRect()	.268
sADMEEntry->GetCheckGlyph()	.269
sADMEEntry->GetDisabledPicture()	.269
sADMEEntry->GetDisabledPictureID()	.270
sADMEEntry->GetID()	.270
sADMEEntry->GetIndex()	.271
sADMEEntry->GetList()	.271

sADMEEntry->GetLocalRect()	.272
sADMEEntry->GetPicture()	.272
sADMEEntry->GetPictureID()	.273
sADMEEntry->GetSelectedPicture()	.273
sADMEEntry->GetSelectedPictureID()	.274
sADMEEntry->GetText()	.274
sADMEEntry->GetTextLength()	.275
sADMEEntry->GetUserData()	.275
sADMEEntry->Invalidate()	.276
sADMEEntry->InvalidateRect()	.276
sADMEEntry->IsActive()	.277
sADMEEntry->IsChecked()	.277
sADMEEntry->IsEnabled()	.278
sADMEEntry->IsInBounds()	.278
sADMEEntry->IsSelected()	.279
sADMEEntry->IsSeparator()	.279
sADMEEntry->LocalToScreenPoint()	.279
sADMEEntry->LocalToScreenRect()	.280
sADMEEntry->MakeInBounds()	.280
sADMEEntry->MakeSeparator()	.281
sADMEEntry->ScreenToLocalPoint()	.281
sADMEEntry->ScreenToLocalRect()	.282
sADMEEntry->Select()	.282
sADMEEntry->SendNotify()	.283
sADMEEntry->SetCheckGlyph()	.284
sADMEEntry->SetDisabledPicture()	.284
sADMEEntry->SetDisabledPictureID()	.285
sADMEEntry->SetID()	.285
sADMEEntry->SetPicture()	.286
sADMEEntry->SetPictureID()	.286
sADMEEntry->SetSelectedPicture()	.287
sADMEEntry->SetSelectedPictureID()	.288
sADMEEntry->SetText()	.288
sADMEEntry->SetUserData()	.289
sADMEEntry->Update()	.290
ADM Help Support	.290
sADMEEntry->GetHelpID()	.290
sADMEEntry->Help()	.291
sADMEEntry->SetHelpID()	.291

Chapter 11 The ADM Hierarchy List Suite 293

About the ADM HierarchyList Suite	.293
Accessing the Suite	.294
ADM Hierarchy Lists and List Entries	.294
ADM Hierarchy List Recipes	.294

Custom ADM Hierarchy Lists295
ADM Hierarchy List Suite Functions296
sADMHierarchyList->DeselectAll()296
sADMHierarchyList->FindEntry()296
sADMHierarchyList->GetActiveEntry()297
sADMHierarchyList->GetActiveLeafEntry()297
sADMHierarchyList->GetDestroyProc()298
sADMHierarchyList->GetDivided()298
sADMHierarchyList->GetDrawProc()299
sADMHierarchyList->GetEntry()299
sADMHierarchyList->GetEntryHeight()300
sADMHierarchyList->GetEntryTextRect()300
sADMHierarchyList->GetEntryWidth()301
sADMHierarchyList->GetExpandedIndex()301
sADMHierarchyList->GetFlags()302
sADMHierarchyList->GetGlobalLeftMargin()302
sADMHierarchyList->GetIndentationWidth()303
sADMHierarchyList->GetInitProc()303
sADMHierarchyList->GetItem()304
sADMHierarchyList->GetLeafIndex()304
sADMHierarchyList->GetLocalLeftMargin()305
sADMHierarchyList->GetLocalRect()305
sADMHierarchyList->GetMask()306
sADMHierarchyList->GetMenuID()306
sADMHierarchyList->GetNonLeafEntryWidth()307
sADMHierarchyList->GetNonLeafEntryTextRectRecursive()307
sADMHierarchyList->GetNotifierData()308
sADMHierarchyList->GetNotifyProc()308
sADMHierarchyList->GetParentEntry()309
sADMHierarchyList->GetTrackProc()309
sADMHierarchyList->GetUserData()310
sADMHierarchyList->GlobalToLocalPoint()310
sADMHierarchyList->GlobalToLocalRect()311
sADMHierarchyList->IndexAllSelectedEntriesInHierarchy()311
sADMHierarchyList->IndexEntry()312
sADMHierarchyList->IndexExpandedEntry()313
sADMHierarchyList->IndexLeafEntry()313
sADMHierarchyList->IndexSelectedEntry()314
sADMHierarchyList->IndexUnNestedSelectedEntriesInHierarchy()315
sADMHierarchyList->InsertEntry()315
sADMHierarchyList->InsertGivenEntry()316
sADMHierarchyList->Invalidate()316
sADMHierarchyList->LocalToGlobalPoint()317
sADMHierarchyList->LocalToGlobalRect()318
sADMHierarchyList->LocalToScreenPoint()318
sADMHierarchyList->NumberOfEntries()319

sADMHierarchyList->NumberOfLeafEntries()	319
sADMHierarchyList->NumberOfSelectedEntries()	320
sADMHierarchyList->NumberOfAllSelectedEntriesInHierarchy()	321
sADMHierarchyList->NumberOfExpandedEntriesInHierarchy()	321
sADMHierarchyList->		
NumberOfUnNestedSelectedEntriesInHierarchy()	322
sADMHierarchyList->PickEntry()	322
sADMHierarchyList->PickLeafEntry()	323
sADMHierarchyList->RemoveEntry()	323
sADMHierarchyList->ScreenToLocalPoint()	324
sADMHierarchyList->SetBackgroundColor()	324
sADMHierarchyList->SetDestroyProc()	325
sADMHierarchyList->SetDestroyProcRecursive()	325
sADMHierarchyList->SetDivided()	326
sADMHierarchyList->SetDividedRecursive()	327
sADMHierarchyList->SetDrawProc()	327
sADMHierarchyList->SetDrawProcRecursive()	328
sADMHierarchyList->SetEntryHeight()	329
sADMHierarchyList->SetEntryHeightRecursive()	330
sADMHierarchyList->SetEntryTextRect()	330
sADMHierarchyList->SetEntryTextRectRecursive()	331
sADMHierarchyList->SetEntryWidth()	331
sADMHierarchyList->SetEntryWidthRecursive()	332
sADMHierarchyList->SetFlags()	333
sADMHierarchyList->SetFlagsRecursive()	333
sADMHierarchyList->SetIndentationWidth()	334
sADMHierarchyList->SetIndentationWidthRecursive()	335
sADMHierarchyList->SetInitProc()	335
sADMHierarchyList->SetInitProcRecursive()	336
sADMHierarchyList->SetLocalLeftMargin()	337
sADMHierarchyList->SetMask()	337
sADMHierarchyList->SetMaskRecursive()	338
sADMHierarchyList->SetMenuID()	339
sADMHierarchyList->SetNonLeafEntryTextRect()	339
sADMHierarchyList->SetNonLeafEntryTextRectRecursive()	340
sADMHierarchyList->SetNotifierData()	340
sADMHierarchyList->SetNotifyProc()	341
sADMHierarchyList->SetNotifyProcRecursive()	342
sADMHierarchyList->SetTrackProc()	343
sADMHierarchyList->SetTrackProcRecursive()	344
sADMHierarchyList->SetUserData()	344
sADMHierarchyList->StartMultipleItemInvalidate()	345
sADMHierarchyList->StopMultipleItemInvalidate()	346
sADMHierarchyList->SwapEntries()	346
sADMHierarchyList->UnlinkEntry()	347

Chapter 12 The ADM Icon Suite 349

About the ADM Icon Suite	349
Accessing the Suite	349
ADM Icons	349
ADM Icon Suite Functions	350
sADMIcon->Create()	350
sADMIcon->CreateFromImage()	351
sADMIcon->Destroy()	351
sADMIcon->GetData()	352
sADMIcon->GetFromResource()	352
sADMIcon->GetHeight()	353
sADMIcon->GetType()	353
sADMIcon->GetWidth()	354
sADMIcon->IsFromResource()	354

Chapter 13 The ADM Image Suite 357

About the ADM Image Suite	357
Accessing the Suite	357
ADM Image Suite Functions	357
sADMImage->BeginADMDrawer()	357
sADMImage->BeginAGMImageAccess()	358
sADMImage->BeginBaseAddressAccess()	358
sADMImage->Create()	359
sADMImage->CreateBitmap()	360
sADMImage->CreateOffscreen()	360
sADMImage->Destroy()	361
sADMImage->EndAGMImageAccess()	361
sADMImage->EndBaseAddressAccess()	362
sADMImage->EndADMDrawer()	362
sADMImage->GetBitsPerPixel()	363
sADMImage->GetByteWidth()	363
sADMImage->GetHeight()	363
sADMImage->GetPixel()	364
sADMImage->GetWidth()	364
sADMImage->SetPixel()	365

Chapter 14 The ADM Item Suite 367

About the ADM Item Suite	367
Accessing the Suite	367
Initializing An Item	367
FloatToText and TextToFloat Functions	371

ADM Help Support.372
ADM Item Suite Functions373
sADMItem->AbortTimer()373
sADMItem->Activate()374
sADMItem->AddItem().374
sADMItem->Create()375
sADMItem->CreateTimer()376
sADMItem->DefaultDraw()378
sADMItem->DefaultFloatToText()378
sADMItem->DefaultTextToFloat()379
sADMItem->DefaultNotify()379
sADMItem->DefaultTrack()380
sADMItem->Destroy()381
sADMItem->Enable()382
sADMItem->GetAllowMath().382
sADMItem->GetAllowUnits()383
sADMItem->GetBackColor().383
sADMItem->GetBestSize()384
sADMItem->GetBooleanValue().384
sADMItem->GetBoundsRect().385
sADMItem->GetChildItem()385
sADMItem->GetCursorID()387
sADMItem->GetDestroyProc()387
sADMItem->GetDialog().388
sADMItem->GetDisabledPicture().388
sADMItem->GetDisabledPictureID()389
sADMItem->GetDrawProc()389
sADMItem->GetFixedValue()390
sADMItem->GetFloatToTextProc()390
sADMItem->GetFloatValue()391
sADMItem->GetFont().391
sADMItem->GetForeColor().392
sADMItem->GetHasRolloverProperty()392
sADMItem->GetHierarchyList()393
sADMItem->GetID()393
sADMItem->GetIntValue().393
sADMItem->GetItemStyle()394
sADMItem->GetItemType()394
sADMItem->GetJustify().395
sADMItem->GetLargeIncrement().395
sADMItem->GetList()396
sADMItem->GetLocalRect().396
sADMItem->GetMask()397
sADMItem->GetMaxFixedValue().397
sADMItem->GetMaxFloatValue().398
sADMItem->GetMaxIntValue().398

sADMIItem->GetMaxTextLength()	398
sADMIItem->GetMinFixedValue()	399
sADMIItem->GetMinFloatValue()	399
sADMIItem->GetMinIntValue()	400
sADMIItem->GetNotifierData()	400
sADMIItem->GetNotifyProc()	401
sADMIItem->GetPicture()	401
sADMIItem->GetPictureID()	402
sADMIItem->GetPluginRef()	402
sADMIItem->GetPopupDialog()	403
sADMIItem->GetPrecision()	403
sADMIItem->GetRolloverPicture()	404
sADMIItem->GetRolloverPictureID()	404
sADMIItem->GetSelectedPicture()	405
sADMIItem->GetSelectedPictureID()	406
sADMIItem->GetSelectionRange()	406
sADMIItem->GetShowUnits()	407
sADMIItem->GetSmallIncrement()	407
sADMIItem->GetText()	408
sADMIItem->GetTextLength()	408
sADMIItem->GetTextToFloatProc()	409
sADMIItem->GetTrackProc()	409
sADMIItem->GetUnits()	410
sADMIItem->GetUserData()	410
sADMIItem->GetWantsFocus()	411
sADMIItem->GetWindowRef()	411
sADMIItem->IgnoreForceRoman()	412
sADMIItem->Invalidate()	412
sADMIItem->InvalidateRect()	413
sADMIItem->IsActive()	413
sADMIItem->IsEnabled()	414
sADMIItem->IsInRolloverState()	414
sADMIItem->IsKnown()	415
sADMIItem->IsVisible()	415
sADMIItem->Known()	416
sADMIItem->LocalToScreenPoint()	416
sADMIItem->LocalToScreenRect()	417
sADMIItem->Move()	417
sADMIItem->RemoveItem()	418
sADMIItem->ScreenToLocalPoint()	419
sADMIItem->ScreenToLocalRect()	419
sADMIItem->SelectAll()	420
sADMIItem->SendNotify()	420
sADMIItem->SetAllowMath()	421
sADMIItem->SetAllowUnits()	421
sADMIItem->SetBackColor()	422
sADMIItem->SetBooleanValue()	422

sADMIItem->SetBoundsRect()	423
sADMIItem->SetCursorID()	423
sADMIItem->SetDestroyProc()	424
sADMIItem->SetDisabledPicture()	425
sADMIItem->SetDisabledPictureID()	425
sADMIItem->SetDrawProc()	426
sADMIItem->SetFixedValue()	427
sADMIItem->SetFloatToTextProc()	428
sADMIItem->SetFloatValue()	428
sADMIItem->SetFont()	429
sADMIItem->SetForeColor()	429
sADMIItem->SetHasRolloverProperty()	430
sADMIItem->SetInRolloverState()	430
sADMIItem->SetIntValue()	431
sADMIItem->SetItemStyle()	431
sADMIItem->SetItemType()	432
sADMIItem->SetJustify()	433
sADMIItem->SetLargeIncrement()	434
sADMIItem->SetLocalRect()	434
sADMIItem->SetMask()	435
sADMIItem->SetMaxFixedValue()	435
sADMIItem->SetMaxFloatValue()	436
sADMIItem->SetMaxIntValue()	436
sADMIItem->SetMaxTextLength()	437
sADMIItem->SetMinFixedValue()	437
sADMIItem->SetMinFloatValue()	438
sADMIItem->SetMinIntValue()	438
sADMIItem->SetNotifierData()	439
sADMIItem->SetNotifyProc()	439
sADMIItem->SetPicture()	440
sADMIItem->SetPictureID()	441
sADMIItem->SetPluginRef()	442
sADMIItem->SetPopupDialog()	442
sADMIItem->SetPrecision()	443
sADMIItem->SetRolloverPicture()	444
sADMIItem->SetRolloverPictureID()	444
sADMIItem->SetSelectedPicture()	445
sADMIItem->SetSelectedPictureID()	445
sADMIItem->SetSelectionRange()	446
sADMIItem->SetSmallIncrement()	447
sADMIItem->SetText()	448
sADMIItem->SetTextToFloatProc()	449
sADMIItem->SetTrackProc()	450
sADMIItem->SetUnits()	451
sADMIItem->SetUserData()	452
sADMIItem->SetWantsFocus()	452
sADMIItem->Show()	453

sADMLItem->ShowUnits()	.453
sADMLItem->Size()	.454
sADMLItem->Update()	.454
sADMLItem->WasPercentageChange()	.455
ADM Help Support	.455
sADMLItem->EnableTip()	.456
sADMLItem->GetHelpID()	.456
sADMLItem->GetTipString()	.457
sADMLItem->GetTipStringLength()	.457
sADMLItem->Help()	.458
sADMLItem->HideToolTip()	.458
sADMLItem->IsTipEnabled()	.458
sADMLItem->SetHelpID()	.459
sADMLItem->SetTipString()	.459
sADMLItem->ShowToolTip()	.460
Chapter 15 The ADM List Suite	461
About the ADM List Suite	.461
Accessing the Suite	.461
ADM Lists and Entries	.461
ADM List Recipes	.462
Custom ADM Lists	.462
ADM List Suite Functions	.463
sADMLList->FindEntry()	.463
sADMLList->GetActiveEntry()	.463
sADMLList->GetDestroyProc()	.464
sADMLList->GetDrawProc()	.465
sADMLList->GetEntry()	.465
sADMLList->GetEntryHeight()	.466
sADMLList->GetEntryTextRect()	.466
sADMLList->GetEntryWidth()	.467
sADMLList->GetInitProc()	.467
sADMLList->GetItem()	.468
sADMLList->GetMask()	.468
sADMLList->GetMenuID()	.469
sADMLList->GetNotifierData()	.469
sADMLList->GetNotifyProc()	.469
sADMLList->GetTrackProc()	.470
sADMLList->GetUserData()	.471
sADMLList->IndexEntry()	.471
sADMLList->IndexSelectedEntry()	.472
sADMLList->InsertEntry()	.473
sADMLList->NumberOfEntries()	.473
sADMLList->NumberOfSelectedEntries()	.474

sADMList->PickEntry()	.474
sADMList->RemoveEntry()	.475
sADMList->SelectByText()	.475
sADMList->SetBackgroundColor()	.475
sADMList->SetDestroyProc()	.476
sADMList->SetDrawProc()	.477
sADMList->SetEntryHeight()	.477
sADMList->SetEntryTextRect()	.478
sADMList->SetEntryWidth()	.478
sADMList->SetInitProc()	.479
sADMList->SetMask()	.480
sADMList->SetMenuID()	.480
sADMList->SetNotifierData()	.481
sADMList->SetNotifyProc()	.481
sADMList->SetTrackProc()	.482
sADMList->SetUserData()	.483

Chapter 16 The ADM List Entry Suite 485

About the ADM List Entry Suite	.485
ADM List Objects and Entries	
ADM Hierarchy List Objects and List Entries	.486
ADM List Entry Suite Functions	.486
sADMListEntry->AbortTimer()	.486
sADMListEntry->Activate()	.487
sADMListEntry->AreChildrenSelected()	.487
sADMListEntry->Check()	.488
sADMListEntry->Create()	.488
sADMListEntry->CreateChildList()	.489
sADMListEntry->CreateTimer()	.489
sADMListEntry->DefaultDraw()	.491
sADMListEntry->DefaultNotify()	.491
sADMListEntry->DefaultTrack()	.492
sADMListEntry->DeleteChildList()	.493
sADMListEntry->Destroy()	.493
sADMListEntry->Enable()	.494
sADMListEntry->EnableChildSelection()	.494
sADMListEntry->ExpandHierarchy()	.495
sADMListEntry->GetBoundsRect()	.495
sADMListEntry->GetChildList()	.496
sADMListEntry->GetDisabledPicture()	.496
sADMListEntry->GetDisabledPictureID()	.497
sADMListEntry->GetEntryItem()	.497
sADMListEntry->GetExpandArrowLocalRect()	.498
sADMListEntry->GetHierarchyDepth()	.498
sADMListEntry->GetID()	.499

sADMListEntry->GetIndex()	.499
sADMListEntry->GetItem()	.500
sADMListEntry->GetList()	.500
sADMListEntry->GetLocalRect()	.500
sADMListEntry->GetPicture()	.501
sADMListEntry->GetPictureID()	.501
sADMListEntry->GetSelectedPicture()	.502
sADMListEntry->GetSelectedPictureID()	.503
sADMListEntry->GetText()	.503
sADMListEntry->GetTextLength()	.504
sADMListEntry->GetUserData()	.504
sADMListEntry->GetVisualHierarchyDepth()	.505
sADMListEntry->HideEntryName()	.505
sADMListEntry->Invalidate()	.506
sADMListEntry->IsActive()	.507
sADMListEntry->IsChecked()	.507
sADMListEntry->IsChildSelectable()	.508
sADMListEntry->IsEnabled()	.508
sADMListEntry->IsEntryNameHidden()	.509
sADMListEntry->IsHierarchyExpanded()	.509
sADMListEntry->IsInBounds()	.509
sADMListEntry->IsSelected()	.510
sADMListEntry->IsSeparator()	.510
sADMListEntry->LocalToScreenPoint()	.511
sADMListEntry->LocalToScreenRect()	.511
sADMListEntry->MakeInBounds()	.512
sADMListEntry->MakeSeparator()	.512
sADMListEntry->ScreenToLocalPoint()	.513
sADMListEntry->ScreenToLocalRect()	.513
sADMListEntry->Select()	.514
sADMListEntry->SendNotify()	.514
ADMListEntry->SetBackgroundColor()	.515
sADMListEntry->SetDisabledPicture()	.515
sADMListEntry->SetDisabledPictureID()	.516
ADMListEntry->SetDividingLineColor()	.517
ADMListEntry->SetEntryItem()	.517
sAMListEntry->SetEntryTextRect()	.518
sAMListEntry->SetFont()	.518
sADMListEntry->SetID()	.518
sADMListEntry->SetPicture()	.519
sADMListEntry->SetPictureID()	.519
sADMListEntry->SetSelectedPicture()	.520
sADMListEntry->SetSelectedPictureID()	.521
sADMListEntry->SetText()	.521
sAMListEntry->SetTextColor()	.522
sADMListEntry->SetUserData()	.522
sADMListEntry->Update()	.523

ADM Help Support523
sADMListEntry->GetHelpID()524
sADMListEntry->Help()524
sADMListEntry->SetHelpID()524
Chapter 17 The ADM Notifier Suite	527
About the ADM Notifier Suite527
Accessing the Suite527
ADM Notifier Functions527
Using ADM Notifier Functions528
ADM Notifier Types528
ADM Notifier Suite Functions533
sADMNotifier->GetDialog()533
sADMNotifier->GetItem()534
sADMNotifier->GetNotifierType()534
sADMNotifier->IsNotifierType()535
sADMNotifier->SkipNextClipboardOperation()535
Chapter 18 The ADM Tracker Suite	537
About the ADM Tracker Suite537
Accessing the Suite537
ADM Trackers537
ADM Tracker Suite Functions538
sADMTracker->Abort()538
sADMTracker->GetAction()538
sADMTracker->GetCharacter()539
sADMTracker->GetModifiers()539
sADMTracker->GetMouseState()539
sADMTracker->GetPoint()540
sADMTracker->GetTime()540
sADMTracker->GetVirtualKey()541
sADMTracker->ReleaseMouseCapture()541
sADMTracker->TestAction()541
sADMTracker->TestModifier()542
Appendix A ADM Folders and Files	543
Files in SDK ADM Folders543
Appendix B ADM Glossary	545

Appendix C	Frequently Asked Questions (FAQ)	549
	Frequently Asked Questions	549
Appendix D	ADM Error Codes	569
API Index		571

Preface

Introduction

This document describes the Adobe Dialog Manager (ADM). ADM is a collection of APIs for displaying and controlling dialogs in a platform-independent way. This document begins with an overview of the window and control architecture, then presents chapters describing how to use ADM with several Adobe products, and continues with individual chapters for each API suite.

Each suite chapter contains a general introduction to the suite, followed by any concepts and structures used by the suite, and then a description of each specific suite function.

Conventions

Whenever an ADM term is followed by the noun “suite” or “object,” it is prefaced with “ADM” and an initial capital letter is used with the term. For example:

“The ADM Basic suite contains a variety of APIs.”

“A pointer provides access an ADM Dialog object.”

But when referring to *instances* of objects (everywhere else), initial capital letters are *not* used. For example:

“The list is then searched for active ADM entries.”

Constants are denoted with a preceding lowercase **k** (e.g., **kADMClippedTextStaticStyle**). The capital letters ADM in a suite name means that the suite is provided by ADM. For more information on the usage of terms, see [Appendix B](#).

By convention, pointers to suites are named as follows:

```
ADMBasicSuite *sADMBasic;
ADMDialogSuite *sADMDialog;
ADMDialogGroupSuite *sADMDialogGroup;
ADMDrawerSuite *sADMDrawer;
ADMEEntrySuite *sADMEEntry;
ADMHierarchyListSuite *sADMHierarchyList;
ADMIconSuite *sADMIcon;
ADMImageSuite *sADMImage;
ADMItemSuite *sADMItem;
ADMListSuite *sADMList;
ADMListEntrySuite *sADMListEntry;
ADMNotifierSuite *sADMNotifier;
```

```
ADMTrackerSuite *sADMTracker;
```

This convention is followed in this document.

Accessing Suites

Each of the suite chapters has a section named “Accessing the Suite” with suite constants and an example of how the suites are acquired. The examples look like this:

```
ADMDialogSuite *sADMDialog;  
error = sSPBasic->AcquireSuite(kADMDialogSuite, kADMDialogSuiteVersion2,  
&sADMDialog);  
if (error) goto . . . //handle error
```

The **sSPBasic** variable in the code above is assigned a pointer when your plug-in loads. This pointer enables access to a data structure that enables access to the suites. Some applications may provide other, more transparent methods for obtaining suites through their own APIs.

Supporting Documents

Other documentation accompanies SDKs. This document describes the ADM API and how to use it. Since ADM is usually loaded as a PICA plug-in when the application is launched, the *Adobe PICA Programmer's Guide and Reference* is also of interest to users of ADM. It is available, along with the SDKs, from <http://partners.adobe.com>.

1

ADM Overview

About Adobe Dialog Manager

The Adobe Dialog Manager (ADM) is a cross-platform API for implementing dialog interfaces for Adobe applications such as Acrobat, Photoshop, Illustrator, and After Effects. This document describes ADM structures and how to access them. You should already be familiar with the concept of dialogs and dialog items.

ADM enables developers to create and manage cross-platform dialogs. Two types of dialogs are supported: modal dialogs and modeless dialogs. The latter dialogs “float” over the host application windows, while the former are displayed and disappear upon conclusion of the user input. With a modal dialog, a user cannot work elsewhere in the application until the dialog is closed. In both cases, ADM supports a wide variety of control types, including basic ones such as buttons and text, and more complicated types such as lists and hierarchy lists. In addition to providing this wide array of custom and standard user interface elements, ADM also provides some very useful behaviors for free, such as tab palettes and docking palettes, and automatically tracking and displaying the correct selection in grouped radio buttons. Finally, ADM provides a consistent Adobe interface and “look and feel.”

ADM is implemented as a PICA plug-in and uses the PICA suites to export its functionality, but this document gives only a brief description of that plug-in architecture. For additional information, please see the *Adobe PICA Programmer’s Guide and Reference*.

Basic ADM functionality is provided using three core function suites: the ADM Basic suite (basic user interactions and utilities), the ADM Dialog suite (creating/managing dialogs), and the ADM Item suite (creating/managing items in a dialog). A number of additional suites provide other behaviors and allow ADM’s functionality to be extended to cover many different custom interfaces. C or C++ interfaces can be used for each suite.

ADM Design

PICA Plug-ins

PICA is an Adobe standard plug-in architecture used by several Adobe Systems applications such as Acrobat, Photoshop, Illustrator, and After Effects. A plug-in is any file containing a computer program and resources that extend the functionality of the host application. PICA provides a common plug-in management core to the host application and a standard interface for plug-ins. In Adobe documentation and header

files, PICA is often referred to as Sweet Pea, SuitePea, SweetPEA, SuiteP, etc.; these terms may all be considered synonymous with PICA.

The ADM application programming interface (API) is exposed to the host and its plug-in's via "suites." A suite is simply a pointer to a data structure that provides an interface to some common object, often a collection of function pointers (e.g., a group of functions to access an ADM Dialog object). Plug-ins can extend the host API by providing their own function suites.

Before they can be used, all suites must be "acquired"; when no longer needed, suites are "released". This mechanism guarantees that the functions are always available to the plug-in.

An acquired suite is actually a pointer to a structure with the suite's function pointers. To call one of the suite functions, the syntax is:

```
sSuite->function();
```

So to use a suite function, you do something like this:

```
SPBasicSuite *sSPBasic = message->basic;
ADMBasicSuite *sADMBasic;

sSPBasic->AcquireSuite(kADMBasicSuite, kADMBasicSuiteVersion2,
&sADMBasic);
sADMBasic->Beep();
sBasic->ReleaseSuite(kADMBasicSuite, kADMBasicSuiteVersion2);
```

The convention used by most SDK's is for suite variables to be global in scope and indicated by a small "s" followed by the suite name—e.g., **sADMBasic**, as shown above.

Typically, the version number parameter that you pass to **AcquireSuite()** should be the version that contains the functions you need and that you know work! All available suite versions are contained in the corresponding ADM header file (e.g., `ADMBasic.h`) so you can include this header in any project you are writing.

NOTE: Do not assume that higher numbered versions of the product are supersets of lower numbered versions— they may not be.

PICA plug-ins are called by the application at certain times. A PICA event is received through the plug-in's main entry point, which is defined as:

```
SPAPI SPErr PluginMain(char *caller, char *selector, void *message);
```

The caller and selector indicate the type of event. The message is a pointer to a structure with any data necessary to handle the event. The ADM message structure always has the following data in it:

```
typedef struct SPMessagesData {

    long SPCheck;          /* kSPValidSPMessagesData if a valid SPMessages */
    struct SPPlugin *self; /* SPPluginRef */
    void *globals;
    struct SPBasicSuite *basic;
```

```
} SPMessageData;
```

Plug-ins might also be called through callbacks they give to some host, such as the application or ADM. In this case, it is the caller's responsibility to specify what information is available and provide enough information for the plug-in to work.

PICA plug-ins are loaded into and unloaded from memory as needed. When a PICA plug-in adds an ADM dialog, it remains in memory until the dialog is disposed of (for PICA version 2.4 and later; earlier versions of PICA require the plug-in to acquire itself in order to remain in memory).

Some SDKs, such as the Adobe Acrobat SDK, provide special code that handles suite acquisition and release automatically so that the programmer doesn't need to worry about these details. See [Chapter 2, "Using ADM with Adobe Acrobat"](#).

ADM Objects in General

ADM user interfaces are built out of ADM user interface objects. These objects include the dialog windows (dialog objects) and the dialog items (item objects) within the windows.

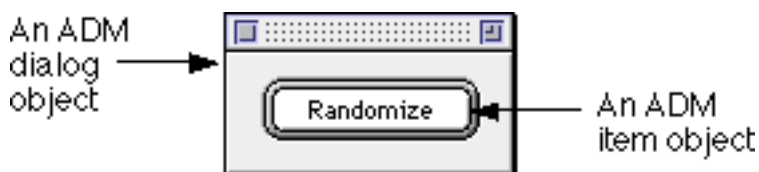


FIGURE 1.1 ADM Objects

A plug-in or application using ADM has access to standard ADM dialog types (modal and non-modal) and items (buttons and other user interface controls). The user interface can be built in code or by using resource definitions. For instance, standard platform resources can be used to define the layout of the UI objects. All objects have *properties* and *events* that determine their default behavior and allow them to be modified or extended. These can also be set in code or via a resource.

ADM has an object-oriented design even though its interfaces are exported as procedural C functions. This is important since many of the properties, behaviors and callback functions of the various types of ADM UI objects (dialogs or dialog items) are the same. Understanding the fundamentals of managing one type of ADM UI object results in understanding how to manipulate other ADM objects as well.

For instance, ADM objects have associated text. For ADM windows this is the window title. For a button, the text is the button title. For an edit text item, the text is the

editable text entered by the user. To access any ADM UI item's text, you can use these two functions:

```
void ASAPI (*SetText)(ADMItemRef inItem, const char* inText);  
void ASAPI (*GetText)(ADMItemRef inItem, const char* inText, ASInt32  
inMaxLen);
```

Some ADM objects need additional support functions or properties. A window object, for instance, has functions to perform operations such as setting the minimum and maximum window size. ADM edit text items have additional functions to support properties such as justification, numeric precision, etc.

The complete ADM object hierarchy looks like this:

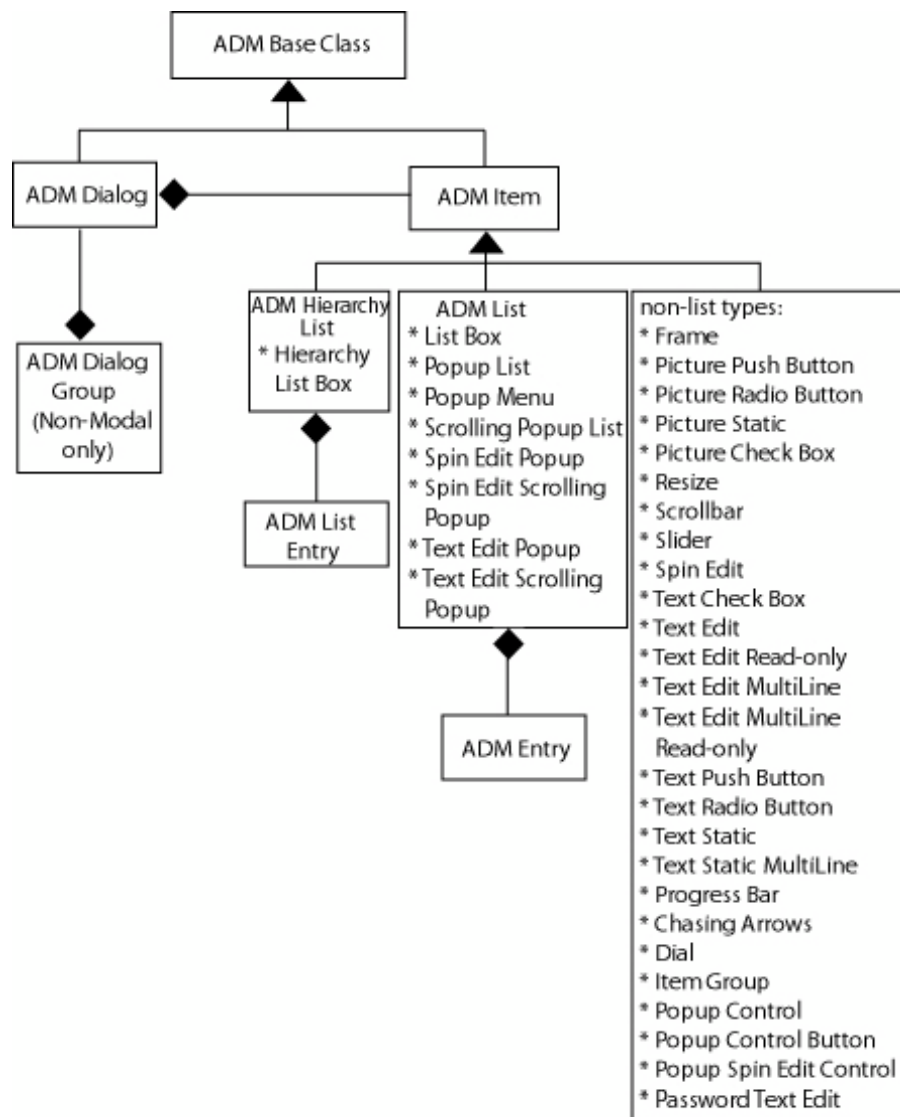


FIGURE 1.2 *The ADM Object Hierarchy*

ADM List objects are owned by List Box, Popup List, Popup Menu, Scrolling Pop List, Spin Edit Popup, Spin Edit Scrolling Popup, Text Edit Popup, and Text Edit Scrolling Popup items. ADM Entry objects are contained by an ADM List.

A Quick Summary of Using ADM

To use ADM, you first use a platform-specific resource editor to add a dialog resource to your plug-in file. At an appropriate point in your plug-in code (likely responding to an application API event), you create a new ADM dialog with either the [sADMDialog-](#)

[>Modal\(\)](#) (this creates a modal dialog) or [sADMDialog->Create\(\)](#) (this creates a non-modal, or floating or tabbed, dialog). You pass this function an initialization function that is called after ADM has loaded the resources and created the dialog. You use this opportunity to set initial values or otherwise customize the dialog's behavior. ADM provides several suites of functions for accessing ADM objects and these are used to perform the initialization.

ADM will then display and handle the dialog for you, processing user events as needed. You will be called to handle certain standard events, such as closing the dialog, and any other events that you request in your init function, such as notification that a button has been pressed. For non-modal dialogs, you call the ADM Dialog suite function [sADMDialog->Destroy\(\)](#) when the dialog is no longer needed. ADM will destroy the dialog and free its resources.

Types

The data types used by ADM are defined in the `ASTypes.h`, `ADMTTypes.h`, `ADMCustomerResource.h`, and `ADMAGMTTypes.h` files.

NOTE: The **AS** prefix is an Adobe-specific convention. Data types with this prefix are used across products.

To ensure platform independence, ADM and other Adobe products use platform-independent type names for some native data types. The following type definitions are from the `ASTypes.h` file.

Types Defined the Same Across Platforms

```
// Integer Types

typedef signed char ASInt8;
typedef signed short ASInt16;
typedef signed long ASInt32;

typedef unsigned char ASUInt8;
typedef unsigned short ASUInt16;
typedef unsigned long ASUInt32;

typedef long ASErr;

// Storage Types

typedef unsigned char ASByte;
typedef ASByte* ASBytePtr;

// Unicode Types
typedef ASUInt16 ASUnicode;

// Pointer Types

typedef void* ASPtr;
```

```
typedef void** ASHandle;

// Fixed Types

typedef long ASFixed;
typedef long ASFract;
typedef float ASReal;

#ifndef _H_ASExpT
typedef struct _t_ASFixedPoint {
    ASFixed h, v;
} ASFixedPoint;

typedef struct _t_ASFixedRect {
    ASFixed left, top, right, bottom;
} ASFixedRect;

typedef struct _t_ASFixedMatrix {
    ASFixed a, b, c, d, tx, ty;
} ASFixedMatrix;

typedef struct _t_ASRealPoint {
    ASReal h, v;
} ASRealPoint;

typedef struct _t_ASRealRect {
    ASReal left, top, right, bottom;
} ASRealRect;

typedef struct _t_ASRealMatrix {
    ASReal a, b, c, d, tx, ty;
} ASRealMatrix;
// ASRGBColor is the same as a Macintosh RGBColor on Macintosh and
// Windows.
typedef struct _t_ASRGBColor {
    unsigned short red, green, blue;
} ASRGBColor;

// AIEvent is the same as a Macintosh EventRecord on Macintosh and
// Windows.
typedef struct _t_ASEvent {
    unsigned short    what;
    unsigned long     message;
    unsigned long     when;
    ASPoint           where;
    unsigned short    modifiers;
} ASEvent;
```

Types Defined Differently Across Platforms

```
// Platform Structures
```

```

// ASBoolean is the same a Macintosh boolean.
typedef unsigned char ASBoolean;

// ASPortRef is the same as a Macintosh GrafPtr.
#if Platform_Carbon
typedef struct OpaqueGrafPtr* ASPortRef;
#else
typedef struct GrafPort* ASPortRef;
#endif

// ASWindowRef is the same as a Macintosh WindowPtr.
#if Platform_Carbon
typedef struct OpaqueWindowPtr* ASWindowRef;
#else
typedef struct GrafPort* ASWindowRef;
#endif

// ASRect is the same size and layout as a Macintosh Rect.
typedef struct _t_ASRect {
    short top, left, bottom, right;
} ASRect;

// ASPoint is the same size and layout as a Macintosh Point.
typedef struct _t_ASPoint {
    short v, h;
} ASPoint;

// ASBoolean is the same a Windows BOOL.
typedef int ASBoolean;

// ASPortRef is the same as a Windows HDC.
typedef void* ASPortRef;

// ASWindowRef is the same as a Windows HWND.
typedef void* ASWindowRef;

// ASRect is the same size and layout as a Windows RECT.
typedef struct _t_ASRect {
    long left, top, right, bottom;
} ASRect;

// ASPoint is the same size and layout as a Windows POINT.
typedef struct _t_ASPoint {
    long h, v;
} ASPoint;

```

Coordinates Using ASRect

The **ASRect** data structure specifies a rectangle of coordinates. Please note, however, that coordinates are between pixels. For example, using coordinates, if you invalidate (see [sADMDialog->Invalidate\(\)](#)) columns 0 - 3 and columns 4 - 6, the pixels

in between will not be re-painted. The periods below represent coordinates and the P's represent pixels. The bold pixels would not be repainted.

. P . P . P . **P** . P . P

. P . P . P . **P** . P . P

Events

There are five events received by all ADM user interface objects. These are listed and described in [Table 1.1](#).

TABLE 1.1 *Events Common to All ADM UI Objects*

Event	When Received
Init	When object is created
Draw	When screen is invalidated or updated
Track	When mouse is over the object
Notify	When the object is hit
Destroy	When the object is disposed of

For most UI objects, you can rely on the default behavior for an event. For instance, when the cursor moves over a text item, ADM will change it to the insert text cursor.

If the behavior of an object at a given event is not what is desired, it can be changed by assigning a new event handler. One event whose behavior you may frequently modify is the Notify event. This is used to check when an object is hit. It is used, for instance, to assign an action to a button click or do special checking on a text entry item.

Properties

There are many properties that are common to all ADM UI objects. These are listed and described in [Table 1.2](#).

TABLE 1.2 *Basic Properties Common to All ADM UI objects*

Properties and Data	Purpose
Type	Defines the general function of the object
Style	Determines the appearance and/or behavior of the object
ID	Numeric reference to the object in its defining space (e.g., its resource number or item number)

TABLE 1.2 Basic Properties Common to All ADM UI objects

Properties and Data	Purpose
Text	Depending on the item, it is usually the title, text value, or the name of an item
Visible	Whether or not the object is visible
Enabled	Whether or not the object is enabled
Active	Whether or not the object is the active item, meaning having keyboard focus (e.g., editable text, or—on Windows—activated by the Enter key).
Known	Whether or not the object is known—an item is in a “known” state if it has a “good” or valid value.
Plug-in	A reference to the plug-in that created the object
UserData	A pointer to any special data assigned to the object when it was created
LocalRect	The size of the object (0,0)-based
BoundsRect	The rectangle of the object in its container’s space. A dialog item is located within a dialog, which is located within the screen bounds.
Properties and Data	Purpose

The first four properties define the object’s function and appearance. Type is the broad category for an object—for example, modal and non-modal dialogs, or popup menus and edit text items. The style property further defines the type of object. As an example, for dialog objects, it indicates whether a modeless dialog is a tab palette or a stand-alone window. ADM UI objects may have one style or many styles. The text associated with an object may be constant, as in a button, or changeable by the user, as with a menu item.

The next four properties of an ADM UI object are state values indicating whether it is visible, enabled, active, and known. If an object is enabled it is usable by the user; if it is disabled, it will have a dimmed appearance and be unusable. If an item is active, it is the focus of current keyboard events. There is only one active dialog item object in a given dialog. On Macintosh computers, only edit text items can be active. On Windows machines, any item can be active, which for non-edit text items means it is the focus of the **Enter** key. For more discussion of these terms, see [Appendix B](#).

Two properties associated with ADM objects allow them to access data without the need for global variables. All ADM items have a reference to the plug-in that created them. This is used when the ADM dialog needs to access a plug-in resource. In addition, when dialog elements are created, a pointer to any custom data is also created. This can point to any type of data structure your dialog needs.

The final two properties, *LocalRect* and *BoundsRect*, define the object's size and location.

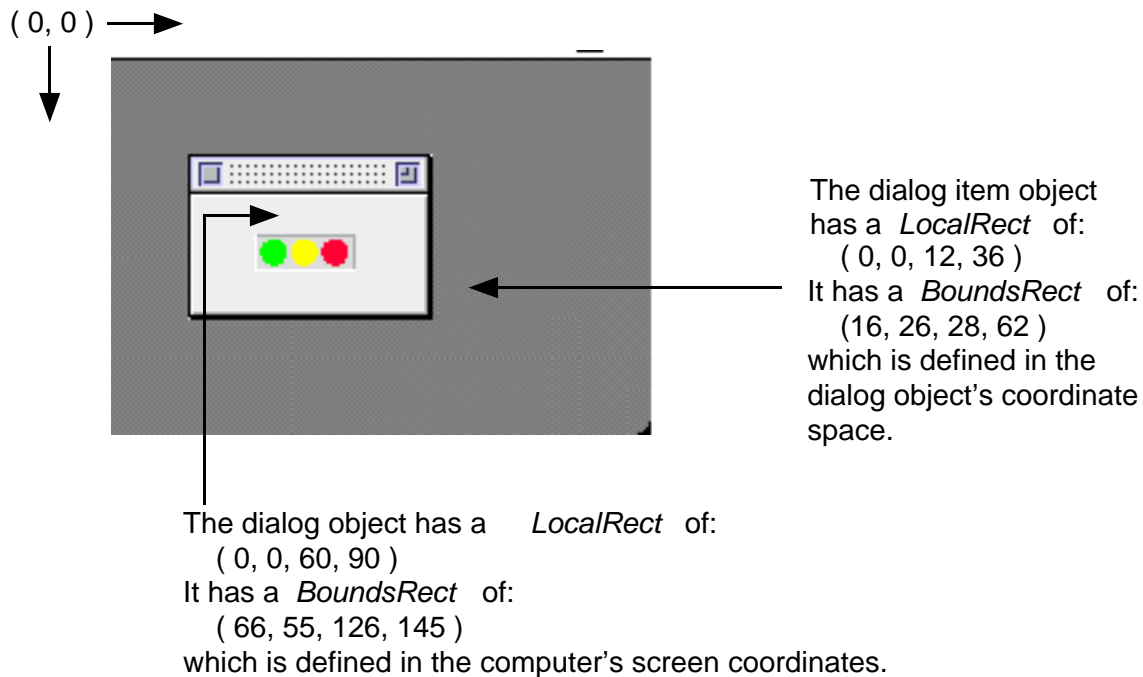


FIGURE 1.3 LocalRects and BoundsRects

The *LocalRect* is the rectangle defining the size of an object in local, (0,0)-based coordinates. The *BoundsRect* is the object's location. It is a rectangle of the same size, but in the object container's coordinate space. The figure above gives an example of this. Note that the coordinates for *BoundsRect* are measured in screen coordinates, which have the (0,0) origin at the upper left hand corner of the screen.

For both *LocalRects* and *BoundsRects* the origin is at the top, left of the rectangle and coordinates increase as they move down and to the right. The origin for tabbed dialogs is not beneath the window title bar, but beneath the tab.

Resources

ADM is designed to simplify the task of creating cross-platform plug-in code for dialogs by largely eliminating the need to support two or more code bases. At the same time, it is intended to support the specific look and feel of its runtime platform. For this reason, dialog resources are created on their host platform while ADM handles how those dialog resources interact with the user. ADM will load and use platform-specific dialog resources correctly. The file `ADMResource.h` defines the constants that are needed when writing ADM dialog resources. The negative IDs are reserved for ADM core implementation, so users should select positive constants for any custom IDs.

On the Macintosh, dialogs are made up of normal **'DLOG'** and **'DITL'** resources. Normal dialog item types can be used for standard controls such as buttons and text items. Item types unique to ADM are implemented as controls defined in [Table 1.5](#). Items that use pictures of some sort can use PICT and icon family resources to define them. ADM will scan for them in that order and use the first resource it finds with the searched for ID.

On Windows, dialog items are window classes. Variations are controlled by class styles. The mapping of Windows window classes and styles to ADM item types and styles is given in [Table 1.4](#). Items that take a picture of some sort can use .bmp and icon resources. ADM will scan for them in that order and use the first resource it finds with the searched for ID.

Setup information for ADM objects on all platforms is given in the sections describing specific item types and in [Table 14.1](#).

The Suites

The ADM manager provides thirteen suites that are used to implement ADM dialogs. The functions of these suites for the 2.8 release of ADM are described in [Chapter 6](#), “The ADM Basic Suite” through [Chapter 18](#), “The ADM Tracker Suite”.

The functions in the suites are standard C style functions. In addition to these, a complete set of C++ wrappers for working with ADM dialogs in as objects is provided in several Adobe SDKs. These wrappers can be found in the IADM (Interface to ADM) directory in the SDK.

The functions for creating and manipulating ADM UI objects are found in a number of header files. The suites that make up ADM's public API are shown in [Table 1.3](#).

TABLE 1.3 *ADM Suites*

Suite	Purpose	Associated Header File
Basic	Provides minimal dialog and resource functionality such as alerts, beeps, resource access, and string utilities.	ADMBasic.h
Dialog	ADM property access functions for dialog objects.	ADMDialog.h
Dialog Group	Functions for grouping dialogs into a docked palette.	ADMDialogGroup.h
Drawer	Functions for implementing custom drawer callbacks.	ADMDrawer.h
Entry	Functions for working with ADM Entry objects.	ADMEEntry.h

TABLE 1.3 **ADM Suites**

Suite	Purpose	Associated Header File
Hierarchy List	Functions for ADM Hierarchy List objects.	ADMHierarchyList.h
Icon	Provides a standard interface to cross-platform picture resources.	ADMIcon.h
Image	Functions for creating off-screen images that can be displayed and manipulated using ADM drawers.	ADMImage.h
Item	ADM property access functions for dialog item objects.	ADMItem.h
List	Functions for ADM List objects.	ADMList.h
List Entry	Functions for ADM Hierarchy List Entry objects.	ADMListEntry.h
Notifier	Functions for implementing custom notifier callbacks.	ADMNotifier.h
Tracker	Functions for implementing custom tracker callbacks.	ADMTracker.h

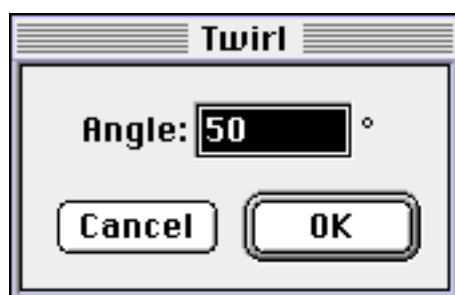
Specific API information is provided in the chapters describing each suite.

ADM Object Specifics

ADM Dialog Objects

ADM Dialog objects are of two types: modal or non-modal (floating). They are further defined by an ADM dialog style. All ADM Dialog objects have a general appearance that complements the main application's user interface, as shown in [Figure 1.4](#).

Modal Dialogs

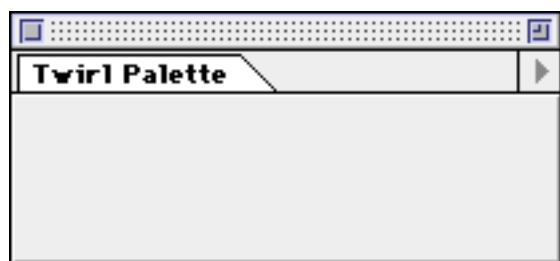


kADMModalDialogStyle

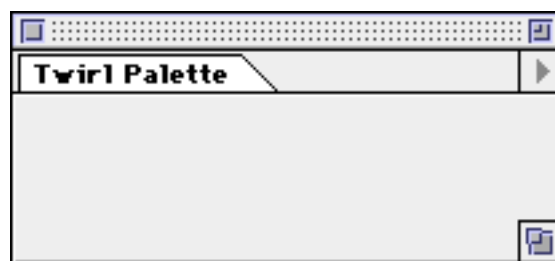


kADMAAlertDialogStyle

Floating Dialogs



kADMTabbedFloatingDialogStyle



kADMTabbedResizingFloatingDialogStyle



kADMFloatingDialogStyle



kADMResizingFloatingDialogStyle

FIGURE 1.4 ADM Dialog Object Types and Styles

Modal dialogs require that the user dismiss the dialog before the host application can be directly used again. They will often have some effect on settings or the application's data upon being dismissed. For an example of how to code a modal dialog, see [Chapter 3, "Using ADM with Adobe Photoshop"](#).

Non-modal dialogs, also called floating dialogs or palettes, will "float" over the main application window and allow the user to switch between the application and the dialog in a highly interactive fashion. If a floating dialog is re-sizable it means the user can grab the platform-specific resize indicator and stretch or shrink the window area. A floating tabbed dialog can be combined or "docked" with others as shown in [Figure 1.5](#). For an example of how to code a non-modal dialog, see [Chapter 4, "Using](#)

ADM with Adobe Illustrator”.

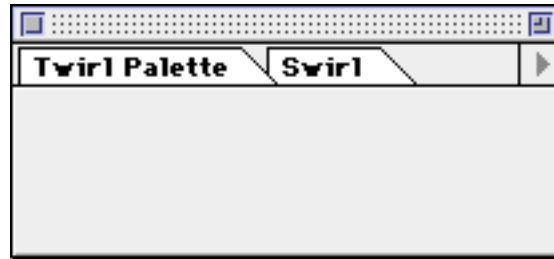


FIGURE 1.5 Combined Tabbed Palettes

Behaviors such as moving a window or combining several tabbed windows are handled automatically by ADM. ADM will handle basic window resizing, but the plug-in will probably need to respond to a resize notification by moving its items or changing their size.

The size of the window is initially set by the size of the window resource. It can also be set via a function at any time. On the Macintosh, ADM modeless dialog windows are specified with 'DLOG' resources using a custom window definition of ID 1991 (WDEF 124 and variation 7). Modal dialog windows are specified with 'DLOG' resources using standard Macintosh dialog resources. On Windows, all ADM dialog windows are standard DIALOG resources. The following code segments show ADM dialog resources for Macintosh and Windows.

```
/* Macintosh */
resource 'DLOG' (16128) {
    {365, 171, 459, 376},
    1991,
    invisible,
    goAway,
    0x0,
    16128,
    "AlignADM Palette"
};

/* Windows */
16000 DIALOG 12, 9, 161, 67
STYLE WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU | WS_MINIMIZEBOX |
WS_MAXIMIZEBOX
CAPTION "Align"
FONT 8, "MS Sans Serif"
{
}
```

The ADM window's style is set at runtime when it is created. These styles are found in `ADMDialog.h` are passed to the dialog creation function.

ADM Dialog objects are created with a plug-in using three calls from the ADM Dialog suite. Two functions, `sADMDialog->Create()` and `sADMDialog->Destroy()`, are for non-

modal (floating) dialogs. To make a non-modal dialog, the plug-in calls the [sADMDialog->Create\(\)](#) function. When the modeless dialog is no longer needed, the plug-in calls the [sADMDialog->Destroy\(\)](#) function. For modal dialogs there is only one function, [sADMDialog->Modal\(\)](#), that is called to create the dialog. Modal dialogs are automatically destroyed when the user dismisses them.

Both the [sADMDialog->Create\(\)](#) and the [sADMDialog->Modal\(\)](#) functions take the same arguments. **inPluginRef** is for the plug-in creating the dialog. **inName** is the name of the dialog window resource. Please note that this is an internal name—not the title of the dialog window. **inDialogID** is the resource number of the platform dialog resource. **inStyle** is one of the ADM dialog style constants in the header files and the examples shown in [Figure 1.4](#). **inInitProc** is a function pointer to a routine that does any initial setup of the dialog, such as positioning it or setting dialog item values. The user **inData** argument is also a pointer, but to a structure you define. It is used to access any data needed by the dialog. **inOptions** provides additional control on dialog creation.

Dialog Item Objects

There are many types of ADM Item objects. Combined with style variations and custom callbacks for drawing, tracking, and notification, you can create just about any dialog appearance and behavior needed. ADM items associated with a dialog are normally created automatically with the dialog. You can manually create and dispose of them in your plug-in. Resource types for all ADM items for each platform are given in [Table 1.4](#) and [Table 1.5](#). [Table 14.1](#) explains how to initialize each item.

ADM items are defined in `ADMItem.h`, as is the function suite used to access them. Constants are used to identify each item type. These constants are listed and described below, along with screen shots showing examples of the different item types. In addition to the standard ADM object properties, all ADM items have a parent dialog and a parent window reference.

kADMFrameType

kADMPictureStaticType

The two simplest ADM Item object types are frames and static pictures, as both are used primarily for visual effects. ADM frames are used to visually group dialog items together. ADM static pictures are used to provide some unchanging visual feedback to the user, such as information about the host program they are using.

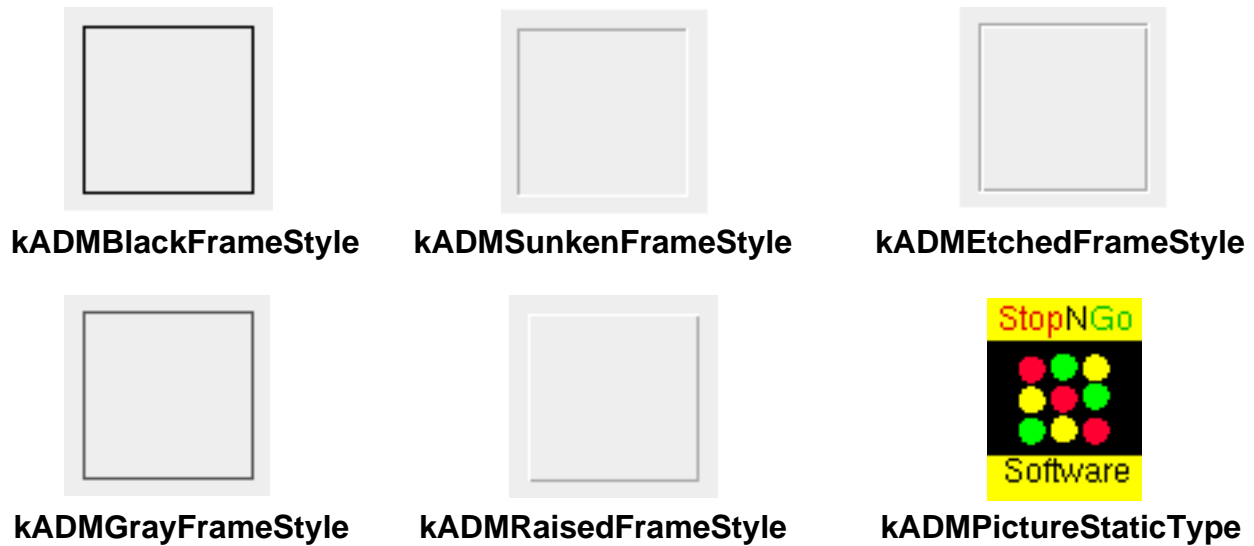


FIGURE 1.6 ADM Frames and Static Pictures

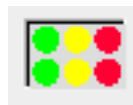
The only information needed to define a frame is its bounding rectangle and its style. These can be set in the dialog resource or created at runtime. To define a frame in the dialog resource, you would create an item with a specific type of frame style and include a bounds rectangle.

A static picture is defined by its bounding rectangle and a picture resource ID. To define a static picture in the dialog resource, you would provide the resource ID and a bounds rectangle.

kADMPicturePushButtonType

kADMTextPushButtonType

Buttons are a common dialog control and ADM offers two types—text buttons and picture buttons. Text buttons display the ADM item text within a rounded rectangle. Picture buttons take three pictures: one for their default state, one for their selected state, and a third for a disabled state. In addition, a button may be the default item, in which case it is enclosed in another rectangle. The **kADMTextPushButtonType** in [Figure 1.7](#) is a default button.

kADMPicturePushButtonType**kADMTextPushButtonType****Enabled****Selected****Disabled****FIGURE 1.7 ADM Button Types**

The information that defines a text push button is its bounding rectangle and its text. These can be set in the dialog resource or created at runtime. A text push button is easily defined using a standard platform button item resource.

A picture push button is defined by its bounding rectangle and the resource IDs for its three pictures. To define a picture push button in the dialog resource, use the values in the platform dialog items chart.

The selected state and disabled state pictures are optional. If resources for these states are not provided, ADM will draw them correctly, offsetting the picture when selected and graying it when disabled.

kADMTextRadioButtonType**kADMPictureRadioButtonType****kADMTextCheckBoxType****kADMPictureCheckBoxType**

Other types of buttons made available by ADM are radio buttons and check boxes. Radio buttons allow the user to choose a single item from a group of options. As with push buttons, radio buttons can be either text buttons or picture buttons. They take the same information as push buttons—either the object's text or up to three pictures for the enabled, selected, and disabled states. Radio buttons that have consecutive ADM item IDs will be automatically grouped together so that only one of the group can be selected.

Check boxes allow the user to set an on/off condition. Check boxes can be of the text or picture type. On Windows, you cannot create a picture check box from platform-specific resources.

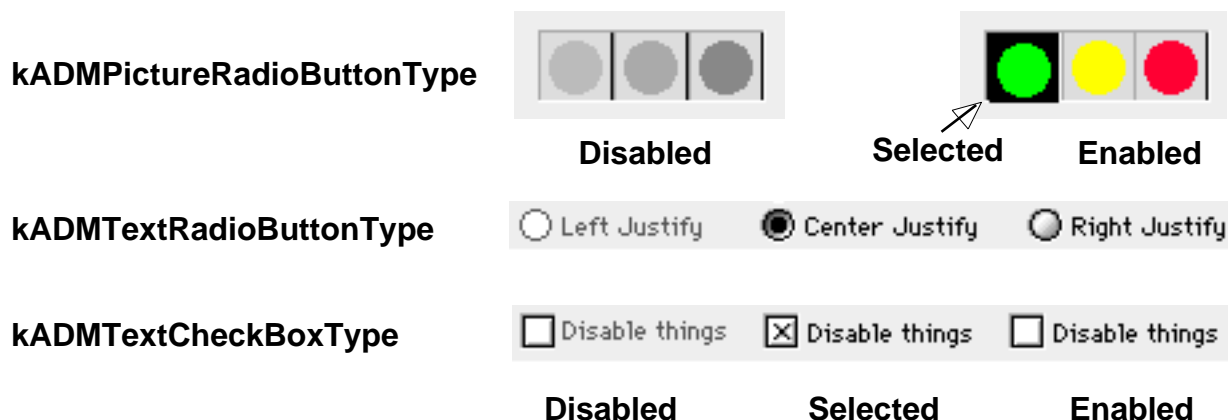


FIGURE 1.8 ADM Radio Buttons and Check Boxes

Both check box and radio button items have a state which indicates whether or not they are selected. This can be set by specifying the boolean value of the dialog item:

```
item = sADMDialog->GetItem(parentDialog, kDisableCheckBox);
sADMItem->SetBooleanValue(item, false);
```

Once the value of an item is set, you don't have to set it again unless you choose to do so. ADM's default behavior will check and uncheck a check box or select and deselect radio buttons in a group. When a radio button in a group is selected, the others in the group will automatically be deselected. Radio buttons with consecutive IDs define a button group.

The value of a radio button or check box can be determined by its boolean value:

```
item = sADMDialog->GetItem(parentDialog, kDisableCheckBox);
if (sADMItem->GetBooleanValue(item))
    // do something
```

Text-based check boxes and radio buttons can be created by supplying a bounds rectangle and the text to be displayed. To define these text-based items in the dialog resource, use the values in the platform-specific dialog items chart.

Picture radio buttons are created by supplying a bounds rect and three picture resource IDs. To define a picture push button in the dialog resource use the values in the platform dialog items chart. Since only the default picture can be defined in the resource on Windows, the disabled and selected pictures should be defined at runtime when the dialog is initialized.

kADMTextEditType

kADMTextEditReadOnlyType

kADMTextStaticType

kADMTextEditMultilineType

kADMTxtEditStaticMultilineType**kADMTxtEditMultilineReadOnlyType****kADMTxtEditScrollingPopUp**

ADM provides a number of text items. Edit text items are used to let the user enter information. Static text items are used to provide information to the user, often as labels for other dialog items. In addition to the two types of text, ADM provides a number of styles, including numeric items and items with multiple lines.

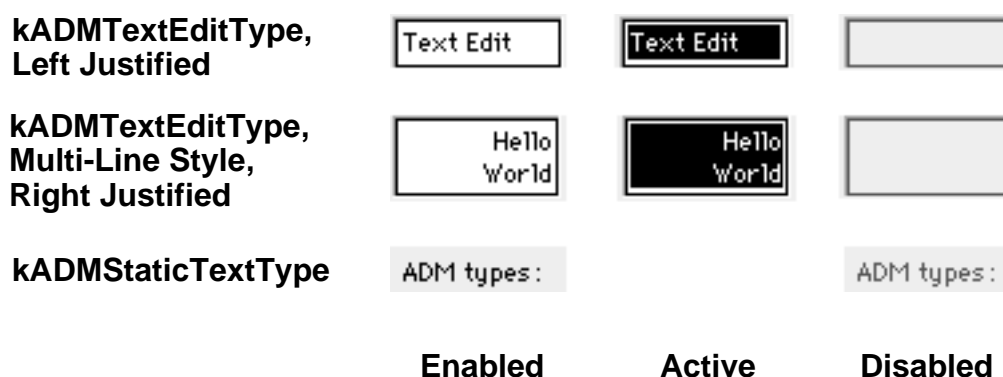


FIGURE 1.9 ADM Text Types and Styles

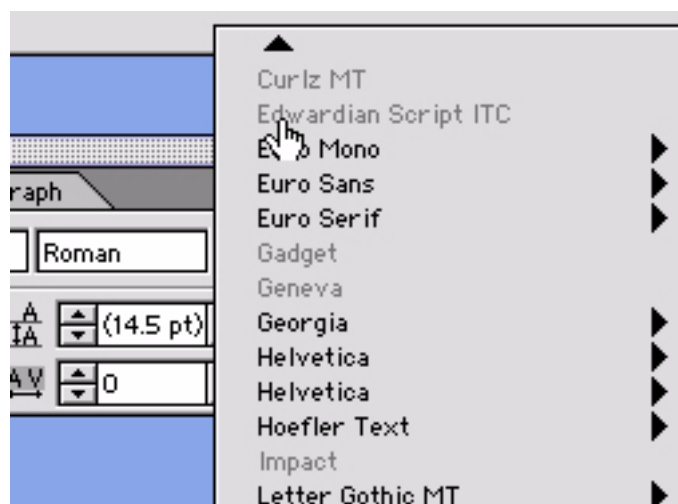


FIGURE 1.10 ADM Text Edit Scrolling Pop-up

The style of a text field can be set in the dialog resource or at runtime using a constant such as the following (see `ADMItem.h`):

```
kADMSingleLineTextEditStyle
kADMNumericTextEditStyle
```

Numeric text fields can have a number of properties that further define the number they can accept, such as valid range. See [ADM Item Numeric Properties](#). Multi-line edit text items will display and scroll multiple lines of text, allowing for carriage returns and automatically wrapping as needed.

All edit text items have a selection range and a maximum length that can be read or set using functions in the ADM Item suite. All text items can have justification set in the dialog resource or at runtime using one of these constants:

```
kADMLeftJustify  
kADMCenterJustify  
kADMRightJustify
```

Numeric text items can have a units value automatically appended to the text. The units for a text field are one of the following:

```
kADMNoUnits  
kADMPointUnits  
kADMInchUnits  
kADMMillimeterUnits  
kADMCentimeterUnits  
kADMPicaUnits  
kADMPercentUnits  
kADMDegreeUnits
```

No text is appended to a numeric text item if it has **kADMNoUnits** for its units property. The units to use can be set at runtime using ADM text item functions.

Static text items are often used as labels for items. A standard behavior for static text labels for edit text items is for the text item to become active when the label is selected. ADM will automatically provide this behavior if the static text label ID immediately precedes or follows the edit text ID.

The text of *any* ADM item can be set and retrieved using two text item functions:

```
char text[65];  
item = sADMDialog->GetItem(parentDialog, kSomeTextItem);  
sADMItem->GetText(item, text, 65);  
updateText(text);  
sADMItem->SetText(item, text);
```

Text items are defined by their bounds rectangle, style, justification, and some text. Their bounds and justification can be set in the dialog resource as indicated by the dialog item resource tables. Other properties of a text item are specified at runtime when the dialog is initialized.

NOTE: The read-only versions of text edit items do not have a platform-specific component.

kADMPopupListType

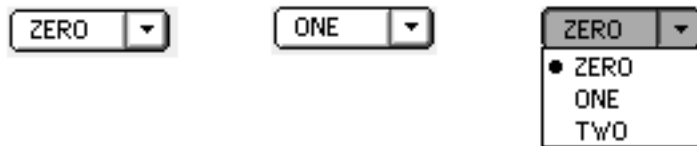
kADMPopupMenuType

kADMScrollingPopupListType

kADMTTextEditPopupType**kADMPopupControlType****kADMPopupControlButtonType****kADMPopupSpinEditControlType****kADMSpinEditScrollingPopupType**

Popup items are a common user interface item in dialogs and ADM provides a number of variations on the basic popup. Popup menus and lists allow the user to choose a single item from a list of options that becomes visible when the item is selected. Popup menus and lists are generally text only, with a standard platform menu resource defining the list of options for the user.

Popup list items display their current setting to the user. Popup menu items appear when the item is selected and you would likely act immediately on the user's selection or display it elsewhere. An ADM text edit popup menu is a combination of a text edit field as described above and a popup menu. The user's popup menu selection will be placed in the text edit field.

kADMPopupListType**kADMPopupMenuType****kADMTTextEditPopupType****Enabled****Disabled****Popped****FIGURE 1.11 ADM Popup Items**

An ADM popup list can be one of two styles. A scrolling popup list can be used on Microsoft Windows machines to add a scrollbar to the popup. On Macintosh computers the scrolling style is ignored.

```
kADMPopupListStyle  
kADMScrollingPopupListStyle
```

An ADM popup menu can be one of two styles. The style variant determines where the popup menu appears. ADM uses this item to create certain item types (e.g., the window menu discussed below and text edit popup items). While you can use popup menus, popup lists are more commonly used.

```
kADMRightPopupMenuStyle  
kADMBottomPopupMenuStyle
```

One common use for popup menus within ADM is to place a menu to the right of tabs in a floating tabbed window. It will be made visible when entries are added to it. Because the origin for tabbed dialogs is beneath the tab and not beneath the window title bar, menu items of this sort have a bounds rectangle with a negative top and 0 for its bottom. You don't need to create this item; ADM will create it automatically for tab-style windows. Its item ID is **kADMMenuItemID**.

Popup text edit items have styles that are a cross between the styles of a popup list and an edit text item:

```
kADMSingleLineEditPopupStyle  
kADMSingleLineEditScrollingPopupStyle  
kADMNumericEditPopupStyle  
kADMNumericEditScrollingPopupStyle
```

If you want to manipulate the individual items in a menu, the ADM menu item is treated as an ADM List object. The list reference for a menu item is obtained using the [sADMItem->GetList\(\)](#) function. There is a suite of functions for performing list operations—the ADM List suite. The items in a menu are actually ADM objects called ADM entries. ADM Entry objects can be enabled or active like any other ADM object. They can also be checked to indicate the current menu value. ADM List objects and ADM Entry objects are discussed more later.

The value of a popup item (the position of its selected item) can be retrieved using the ADM List and ADM Entry suites. You get the active entry in the list and then get the index of the entry.

```
ADMItemRef item = sADMDialog->GetItem(parentDialog, kSomeMenuItem);  
ADMListRef list = sADMItem->GetList(item);  
ADMEEntryRef entry = sADMList->GetActiveEntry(list);  
ASInt32 selection = sADMEEntry->GetIndex(entry);
```

To get the name, you use the [sADMEEntry->GetText\(\)](#) function instead.

Menu items are defined by their bounds rectangle, a menu resource ID, and a style. The bounds rectangle and style of popup items are defined in a resource as indicated in the item resource tables. The menu resource ID of the popup menu's list is specified at runtime when the dialog is initialized. On both Macintosh and Windows, the menu resource type is **'MENU'**.

kADMSpinEditType**kADMSpinEditPopupType****kADMSpinEditScrollingType**

A variation of a text edit item is a spin edit item. Spin edit items provide arrows to increase and decrease their value without typing. A further variation is a spin edit popup item, which adds a popup menu to the spin edit item. Spin edit items have many of the same properties as edit text items, such as justification. They are inherently numeric items and have those properties as well.

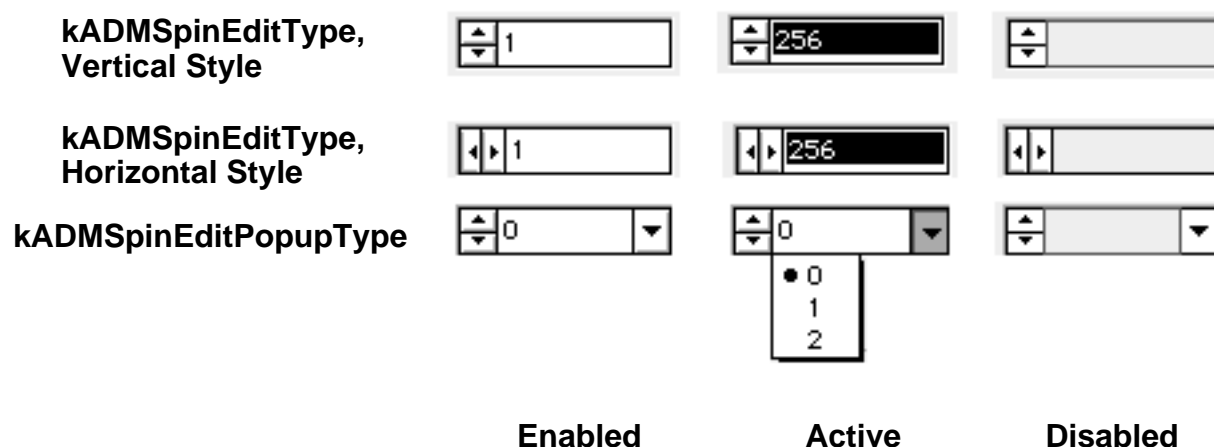


FIGURE 1.12 ADM Spin Edit Items

Spin edit items can have either horizontal or vertical arrows as specified by their style:

```
kADMVerticalSpinEditStyle
kADMHORIZONTALSpinEditStyle
```

Spin edit popup items also have the scrolling style variant of popup lists. This affects the behavior on Windows machines only:

```
kADMVerticalSpinEditPopupStyle
kADMVerticalSpinEditScrollingPopupStyle
kADMHORIZONTALSpinEditPopupStyle
kADMHORIZONTALSpinEditScrollingPopupStyle
```

The rate at which a spin edit control changes the number in its edit field is controlled by its small increment value.

The value of a spin edit item can be retrieved in one of two ways. You can get its value or its text. To get the text, you use the [sADMDialog->GetText\(\)](#) function. To get the value you use the appropriate get value function on the spin item. For instance, to get an integer value you would use:

```
ASInt32 selection;
item = sADMDialog->GetItem(parentDialog, kSomeMenuItem);
selection = sADMItem->GetIntValue(item);
```


A spin edit item is defined by a bounds rectangle and a style, which can be defined in the dialog item resource. When the dialog is initialized the other properties of the item, such as its value and justification, can be defined.

kADMScrollbarType

kADMSliderType

Scrollbars and sliders allow the user to select from a range of values with a graphic interface. The relative position of the current value within the range is indicated by the position of the item's "thumb"—the triangle on the slider and the rectangle within the scrollbar. The item's value can be changed by dragging the thumb. A scrollbar's value can also be changed using the arrows at its ends.

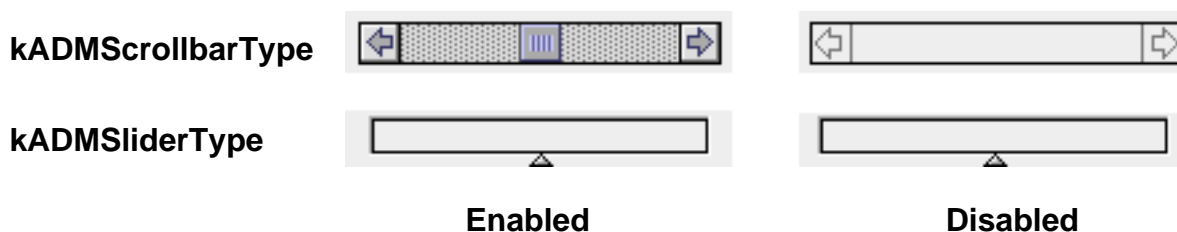


FIGURE 1.13 *ADM Scrollbars and Sliders*

The rate at which a scrollbar item changes its value is controlled by its large and small increment values. The small increment is used when the arrows are clicked and the large increment when the user clicks inside the scrollbar. The value of a slider or scrollbar item can be retrieved using the appropriate get value function on the item. See [ADM Item Numeric Properties](#).

Scrollbar and slider items are defined by their bounds rectangle and a range. The bounds rectangle is specified in the dialog item resource. Their other properties, including their range and large and small increments, are defined at runtime when the dialog is initialized.

kADMListBoxType

kADMHierarchyListType

kADMHierarchyListBoxType

List boxes display a list of options and allow the user to select one or more of them. Their current selection is indicated to the user by inverting the items. If more items are in the list than can be displayed, a scroll bar allows the user to navigate the list. While lists are often text only, they may include graphical information such as a color preview or icon. The display of pictures is handled automatically. More complex lists are created by overriding the list drawing routine.

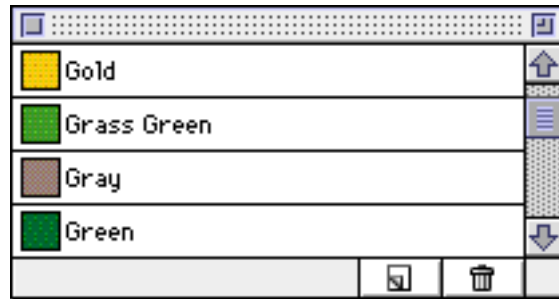


FIGURE 1.14 An ADM List Item with New and Delete Buttons

ADM provides a number of variations on the basic text item list. A list box can allow only a single item to be selected or allow multiple items to be selected. A list can also be created with or without dividing lines between objects. These options are expressed using flags that are ORed together:

```
/* List box styles */
typedef long ADMListBoxStyle;
#define KADMMultiSelectStyle          (1L<<0)
#define KADMDividedStyle              (1L<<2)
#define KADMEntryTextEditableStyle    (1L<<3)
```

Some combinations of these ADMListBox style options are:

```
#define KADMSingleSelectListBoxStyle    0
#define KADMMultiSelectListBoxStyle     (KADMMultiSelectStyle)
#define KADMMultiSelectNewDeleteListBoxStyle
    (KADMMultiSelectStyle|KADMNewDeleteStyle)
#define KADMSingleSelectDividedListBoxStyle (KADMDividedStyle)
#define KADMMultiSelectDividedListBoxStyle
    (KADMMultiSelectStyle|KADMDividedStyle)
#define KADMMultiSelectNewDeleteDividedListBoxStyle
    (KADMMultiSelectStyle|KADMNewDeleteStyle|KADMDividedStyle)
```

The list item is actually a container object for a list and its entries. Each item in a list is an ADM object called an ADM Entry object. ADM Entry objects can be enabled or selected like any other ADM object. They can have special draw functions for custom displays. ADM entry items are also used by ADM popup items. Also available is a suite of functions for performing list operations such as controlling a list's appearance and indexing through its entries.

How you retrieve the list selection depends on the list style. In general, you get the selected entry reference or references and then use the reference to obtain specific information. The code below shows how you might get the selection values from a multi-selection list.

```
ASInt32 selectedCount = sADMList->NumberOfSelectedEntries(theList);
for (i = 0; i < selectedCount; i++) {
    ADMEntryRef theEntry = sADMList->IndexSelectedEntry(theList, i);
    // do something to the entry
```

```
ASInt32 index = sADMEEntry->GetIndex(theEntry);
...
}
```

List box items are defined by their bounds rectangle and a style. The bounds rectangle and style of popup items are defined in a resource as indicated in the item resource tables. Lists can be filled automatically by assigning a menu ID at runtime. Other initialization is also done at runtime when the dialog is initialized.

kADMPProgressBarType

An ADM progress bar indicates that a lengthy operation is occurring. This item uses a **CTL** resource on Macintosh and can be created programmatically on Windows.

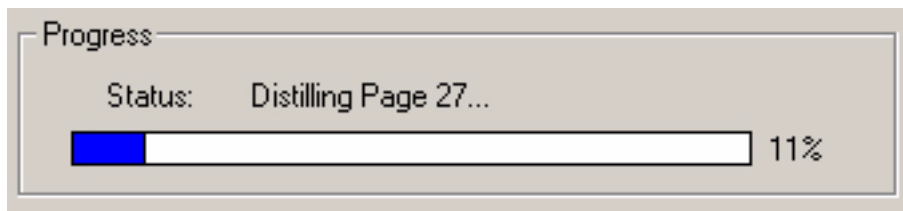


FIGURE 1.15 An ADM Progress Bar

kADMChasingArrowsType

ADM chasing arrows indicate through a simple animation that a background process is in progress. These are available only on the Mac.

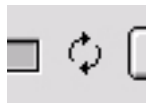


FIGURE 1.16 ADM Chasing Arrows

kADMDialType

An ADM dial is used for calibration. To initialize, you must set its initial value, maximum value, and minimum value.

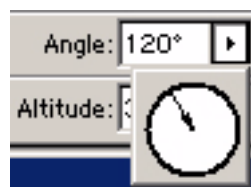


FIGURE 1.17 An ADM Dial

kADMItemGroupType

An ADM item group is a collection of individual items. Item groups make it easier to write notification and tracker callbacks since multiple items are dealt with as though they are a single item. Item groups have no physical representation—they're simply an organizational grouping so are not defined by any specific platform resource. All items respond to single function calls to the group. See [ADM Item Groups](#).

kADMUserType

ADM User and ADM Custom items are used indirectly together to extend ADM with completely new items. A plug-in that provides a custom ADM item uses an item of type **kADMUserType** as a foundation and customizes its behavior. Custom items are discussed more completely in [Custom Item Types](#).

kADMResizeType

The last ADM individual item type is a resize item. This item is created automatically by ADM when a resizable dialog is created and will display the platform's window resize item. Notification of a window being resized is if a notifier handler function is assigned to this item. This function would handle resizing or repositioning items in the dialog. See [Using Resizable Windows](#) for more information.

Composite Items

Some ADM items are actually two or more ADM items composited together. These are list items, spin edit items, spin edit popup items, and text edit popup items. The normal ADM item reference to such an item is to the composite object. The components, or children, of the item can be accessed and then used like any other ADM item—for instance, setting a custom notifier callback function.

The children of a composite item are accessed using the [sADMItem->GetChildItem\(\)](#) function, which is passed a **ChildID** argument. The **ChildIDs** for each composite item are defined in the file `ADMItem.h`. For instance, a list item has these children:

```
typedef enum
{
    kADMListBoxScrollbarChildID = 1,
    kADMListBoxDummyChildID = 0xFFFFFFFF
}
ADMListBoxChildID;
```

ADM Item Groups

If a composite item is not available, an ADM item group allows you to collect a number of items together that need to respond to calls as a group. For example, you might have five items that all need to be enable or disabled simultaneously. Once those items belong to a group, you just need to enable/disable the group.

This is not true of geometrical containment. Item groups really don't have any physical manifestation; they are simply a way of logically grouping items.

ADM Item Numeric Properties

ADM items often have a numeric value. A number of properties can be used to control this value, providing automatic bounds checking or feedback. These properties are **type**, **precision**, **range**, and **increment**.

The numeric **type** refers to how the value is set and retrieved. The valid types are boolean, integer, fixed, and float. Not all item types have these numeric types; for instance, a check box has only a boolean value, while a slider can have any of them. Values are accessed using get and set functions for the type of data desired—for instance, `sADMItem->GetFixedValue()` or `sADMItem->GetMinIntValue()`. The data type of an item will be typecast by the function used to access it. For instance, if the boolean value of a checkbox is retrieved with `sADMItem->GetFloatValue()`, it will be returned as either `0.0` or `1.0`.

The **precision** property of an item refers to how many digits follow the decimal point. Values of an item are automatically limited to the defined precision.

All items except boolean items can have an assigned **range** that sets upper and lower limits on the values that can be assigned to it. ADM automatically confines the value to this specified range in one of two ways. For text edit items and spin edit items, a note alert will appear informing the user of the valid range if an illegal value is entered. The value will then be floored or ceilinged to bring it into range. For sliders and scrollbars, the range is used to calibrate the dialog item. The minimum range value corresponds to the item value when the thumb is in the leftmost position; the maximum when the thumb is in the rightmost position. The range values are accessed using get and set functions for the type of data desired, for instance, `sADMItem->GetMinIntValue()` or `sADMItem->SetMaxFloatValue()`.

You can set the rates at which scrollbar and spinner item values change by setting their **increment** properties. There are small and large increments. The small value is added to or subtracted from the value when an arrow component of the item is clicked once. The large increment is used only by scrollbar items and is added to or subtracted from the item value when the user clicks above or below the thumb inside the scrollbar. The increment values are accessed using get and set functions for the size of increment, for instance, `sADMItem->SetSmallIncrement()`.

NOTE: Increments are integers and are always in the specified units for an item.

For information on text<-->float conversions, see [FloatToText](#) and [TextToFloat Functions](#) in [The ADM Item Suite](#).

ADM Lists and Entries

ADM items based on a list of choices include list boxes, popup lists, popup menus, spin edit popups, and text edit popups. They are all accessed in the same way: as lists of entries. There are two suites of functions that are used to access the list and entry objects, the ADM List suite and ADM Entry suite. The ADM List suite basically lets you access ADM entries. With it you can add and remove entries, iterate through the existing ones, and control the list's entries' height and width. Once you have used the

ADM List suite to access an individual entry, you can use the ADM Entry suite to modify its properties. ADM entries are similar to other ADM UI objects, having properties such as an ID and text, and states like enabled and active.

Entries do not have these standard properties: plug-in, type, style, visible state. They have these additional properties: parent list; index; selected, checked and separator states. The index is the position of the entry in list. The selected state indicates the user has selected the item (others may be selected in the case of a multi-select list). The checked state indicates that a check mark appears to the left of the entry. The separator indicates that the item is a non-selectable item used to break a list into groups of entries. If an entry has an assigned picture, it will automatically be drawn to the left of the text. In addition, an entry's event handler routines cannot be overridden. Special event handling is done by the parent list.

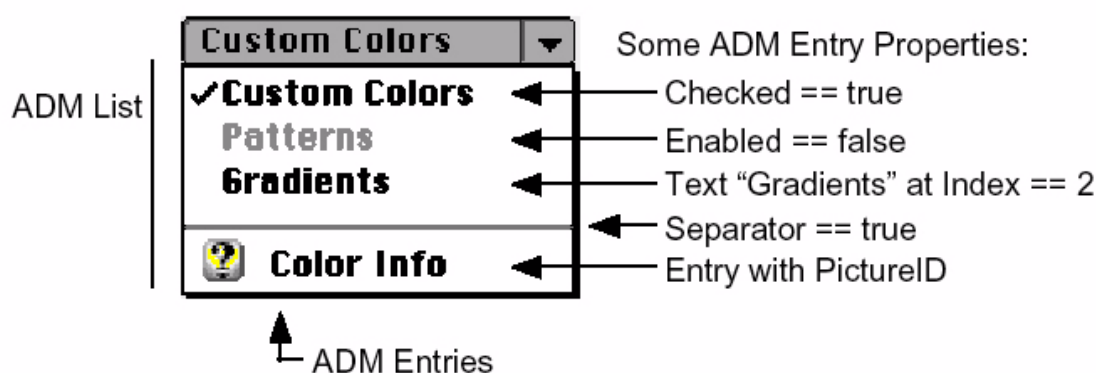


FIGURE 1.18 ADM List and Entry Objects

To get the list object for an item, you use the [sADMItem->GetList\(\)](#) function. Once this is done you can use the ADM List and ADM Entry suites' functions to modify it.

An item's list can be initialized by repeatedly creating entries with the [sADMList->InsertEntry\(\)](#) function and then using the [sADMEEntry->SetText\(\)](#) function to set the new entry's text:

```
for (index = 0; index < kNumberEntries; index++) {
    char menuText[255];
    ADMEEntryRef entry = sADMList->InsertEntry(theItemList, index);
    sBasic->GetIndexString(thePlugin, 16000, index, menuText, 255);
    sADMEEntry->SetText(entry, menuText);
}
```

or more quickly by assigning it a menu resource ID:

```
sADMList->SetMenuID(theItemsList, gPlugInRef, 16000, "Choices");
```

In this case the list items are set corresponding to the items already created in the resource.

Iterating through a list's items is done in a similar fashion to the example given under the [kADMListItem](#) description.

NOTE: List indices are 0-based.

ADM Hierarchy Lists and List Entries

Similarly, the ADM Hierarchy List suite allows you to access ADM Hierarchy List objects and ADM List Entry objects. Since an ADM Hierarchy List object is an extended property of a standard ADM Item object, this suite lacks many of the functions common to ADM objects; however, you can access the a hierarchy list's ADM item and do common operations on it. Using functions in this suite, you can initialize the hierarchy list, and you can create, destroy, customize, and iterate through the ADM list entries of a hierarchy list. The Hierarchy List suite is used in conjunction with the ADM List Entry suite to further access list related information.

NOTE: The relationship between ADM Hierarchy List objects and ADM List Entry objects is the same as that between ADM List objects and ADM Entry objects—that is, list entries are the elements of a hierarchy list. Note that list entries themselves may be hierarchy lists with list entry children of their own.

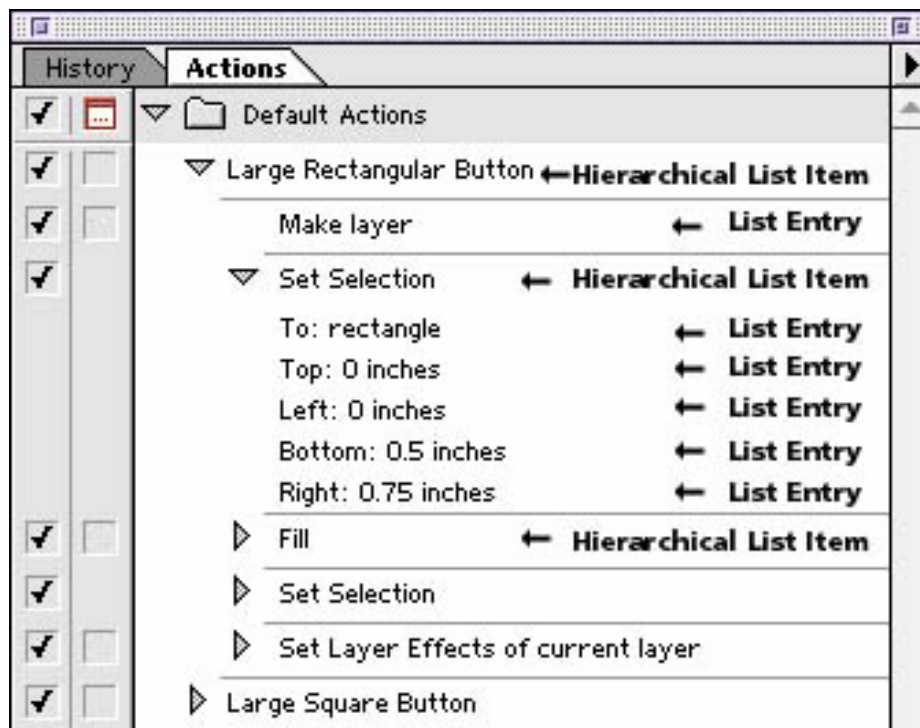


FIGURE 1.19 ADM Hierarchy List and List Entry Objects

NOTE: List indices are 0-based.

Macintosh and Windows ADM Item Resource Lists

Table 1.4 and Table 1.5 list the resource information needed to define ADM items in their native resource formats.

Windows ADM items are defined by the dialog item window class and style which map to an ADM item and style. The item values are set at runtime using the ADM Item suite functions.

NOTE: (0) In the resource file, set the item name to the picture ID to use.

On Macintosh computers, ADM items are created using a dialog item list resource (**DITL**). Simple text-based items such as text push buttons, edit text, etc., can be made using standard Macintosh dialog items. Others are indicated using control items (**CNTL**) with the appropriate **CDEF** and variation (or ProcID, which CDEF * 16 + variation). The variation and other values can also be set at runtime.

NOTE: (1) On Windows plug-ins, these values cannot be set in the resource, but must be set at runtime. Because of this, you may want to set them at runtime on both platforms. (2) On Windows, this can be set in a resource for single line text edit items only.

TABLE 1.4 *Windows ADM Items*

ADM Item Type	ADM Style	Windows Class Name	Window Style
ADM Frame	kADMSunkenFrameStyle	"Button"	BS_GROUPBOX
	kADMBlackFrameStyle	"Static"	SS_BLACKRECT
	kADMGrayFrameStyle	"Static"	SS_GRAYRECT
	kADMRaisedFrameStyle	"Static"	SS_WHITERECT
	kADMBlackFrameStyle	"Static"	SS_BLACKFRAME
	kADMGrayFrameStyle	"Static"	SS_GRAYFRAME
	kADMRaisedFrameStyle	"Static"	SS_WHITEFRAME
	kADMSunkenFrameStyle	"Static"	SS_ETCHEDHORZ
	kADMSunkenFrameStyle	"Static"	SS_ETCHEDVERT
	kADMSunkenFrameStyle	"Static"	SS_ETCHEDFRAME
	ADMFrameStyle	"ADM Frame Type"	
ADM List Box		"Listbox"	
	ADMListBoxStyle	"ADM List Box Type"	
	ADMListBoxStyle	"ADM Hierarchy List Box Type"	

TABLE 1.4 Windows ADM Items

ADM Item Type	ADM Style	Windows Class Name	Window Style
ADM Picture Push Button (0)	item name == MAKEINTRESOURCE(pictureID)	"ADM Picture Push Button Type"	0
ADM Picture Radio Button (0)	item name == MAKEINTRESOURCE(pictureID)	"ADM Picture Radio Button Type"	0
ADM Picture Static (0)	item name == MAKEINTRESOURCE(pictureID)	"Static"	SS_Bitmap
	item name == MAKEINTRESOURCE(pictureID)	"Static"	SS_Icon
	item name == MAKEINTRESOURCE(pictureID)	"Static"	SS_ENHMETAFILE
	item name == MAKEINTRESOURCE(pictureID)	"ADM Picture Static Type"	0
ADM Popup List		"Combobox"	CBS_DROPDOWNLIST
		"ADM Popup List Type"	0
ADM Popup Control		"ADM Popup Control Type"	0
ADM Popup Control Button		"ADM Popup Control Button Type"	0
ADM Popup Spin Edit Control		"ADM Popup Spin Edit Control Type"	0
ADM Popup Menu	ADMPopupMenuStyle	"ADM Popup Menu Type"	0
ADM Resize		"ADM Resize Type"	0
ADM Scrollbar		"Scrollbar"	0
		"ADM Scrollbar Type"	0
ADM Scrolling Popup List		"Combobox"	CBS_DROPDOWNLIST WS_VSCROLL
		"ADM Scrolling Popup List Type"	0
ADM Slider	ADMSliderStyle	"MSCTls_Trackbar32"	0
		"ADM Slider Type"	0
ADM Spin Edit	ADMSpinEditStyle	"ADM Spin Edit Type"	

TABLE 1.4 Windows ADM Items

ADM Item Type	ADM Style	Windows Class Name	Window Style
ADM Spin Edit Popup	kADMSingleLineEditPopupStyle	“Combobox”	CBS_DROPDOWN
	ADMSpinEditStyle	“ADM Spin Edit Popup Type”	
ADM Spin Edit Scrolling Popup	kADMSingleLineEditPopupStyle	“Combobox”	CBS_DROPDOWNLIST WS_VSCROLL
	ADMSpinEditPopupStyle	“ADM Spin Edit Scrolling Popup Type”	
ADM Text Check Box		“Button”	BS_CHECKBOX
		“Button”	BS_AUTOCHECKBOX
		“Button”	BS_3STATE
		“Button”	BS_AUTO3STATE
		“ADM Text Check Box Type”	0
ADM Text Edit	kADMLeftJustify	“Edit”	ES_LEFT
	kADMCenterJustify	“Edit”	ES_CENTER
	kADMRightJustify	“Edit”	ES_RIGHT
	kADMNumericEditStyle (Auto sets if you call SetXValue())	“Edit”	ES_NUMBER
	ADMTextEditStyle	“ADM Text Edit Type”	
ADM Password Text Edit		“ADM Text Edit Type”	ES_PASSWORD
ADM Text Edit Scrolling Popup	kADMSingleLineEditPopupStyle	“Combobox”	CBS_DROPDOWN WS_VSCROLL
	ADMTextEditPopupStyle	“ADM Text Edit Scrolling Popup Type”	
ADM Text Edit Multi Line	kADMNumericEditStyle (Auto sets if you call SetXValue())	“Edit”	ES_MULTILINE
		“ADM Text Edit Multi Line Type”	0

TABLE 1.4 Windows ADM Items

ADM Item Type	ADM Style	Windows Class Name	Window Style
ADM Text Edit Popup	kADMSingleLineEditPopupStyle	“Combobox”	CBS_DROPDOWN
	ADMTextEditPopupStyle	“ADM Text Edit Popup Type”	
ADM Text Push Button	Default	“Button”	BS_DEFPUSHBUTTON
		“Button”	BS_PUSHBUTTON
		“Button”	BS_USERBUTTON
		“Button”	BS_OWNERDRAW
		“ADM Text Push Button Type”	0
ADM Text Radio Button	ADMRadioButtonStyle	“Button”	BS_RADIOBUTTON
		“Button”	BS_AUTORADIOBUTTON
		“ADM Text Radio Button Type”	0
ADM Text Static	kADMLeftJustify	“Static”	SS_LEFT
	kADMCenterJustify	“Static”	SS_CENTER
	kADMRightJustify	“Static”	SS_RIGHT
	kADMLeftJustify	“Static”	SS_LEFTNOWORDWRAP
	kADMLeftJustify	“Static”	SS_SIMPLE
		“ADM Text Static Type”	0
ADM Text Static Multi Line		“Edit”	ES_READONLY
		“ADM Text Static Multi Line Type”	0
ADM User		“ADM User Type”	0
ADM Dial		“ADM Dial Type”	0
ADM Tabbed Menu	Deprecated in V2.8—do not use		
ADM Custom		<custom item name>	““

TABLE 1.5 Macintosh ADM Items

ADMItem Type	Mac Dialog Item	Mac CNTL Resource Settings		Additional Fields			
		CDEF Res ID	Variation	Value	Min	Max	Other
ADM Frame	User Item Control Item	1000	ADMFrameStyle				Control title is the group name (1)
ADM List Box	Control Item	1010	ADMListBoxStyle		MenuID =MenuResID (1)		
ADM Picture Check Box	Control Item	1023	ADMPictureButtonStyle	PictureID	PictureSelectedID (1)	PictureDisabledID (1)	
ADM Picture Push Button	Control Item	1020	ADMPictureButtonStyle	PictureID	PictureSelectedID (1)	PictureDisabledID (1)	
ADM Picture Radio Button	Control Item	1021	ADMPictureButtonStyle	PictureID	PictureSelectedID (1)	PictureDisabledID (1)	
ADM Picture Static	Icon Item Picture Item Control Item	1022	0	PictureID	PictureSelectedID (1)	PictureDisabledID (1)	
ADM Popup List	Control Item	63	0		MenuID =MenuResID (1)		
ADM Popup Control	Control Item	1055	0	ADMJustify (1) IntValue (1)	IntMin (1)	IntMax (1)	
ADM Popup Control Button	Control Item	1056	0	ADMJustify (1) IntValue (1)	IntMin (1)	IntMax (1)	
ADM Popup Spin Edit Control	Control Item	1057	0	ADMJustify (1) IntValue (1)	IntMin (1)	IntMax (1)	

TABLE 1.5 *Macintosh ADM Items*

ADMItem Type	Mac Dialog Item	Mac CNTL Resource Settings		Additional Fields			
		CDEF Res ID	Variation	Value	Min	Max	Other
ADM Popup Menu	Control Item	1030	ADMPopupMenuStyle		MenuID =MenuResID (1)		
ADM Resize	Control Item	1040	0				
ADM Scrollbar	Control Item	1	0	IntValue (1)	IntMin (1)	IntMax (1)	
ADM Scrolling Popup List	Control Item	1031	0				
ADM Slider	Control Item	1050	0	IntValue (1)	IntMin (1)	IntMax (1)	
ADM Spin Edit	Control Item	1060	ADMSpinEditStyle	ADMJustify (1)			
ADM Spin Edit Popup	Control Item	1061	ADMSpinEditPopupStyle	ADMJustify (1)	MenuID =MenuResID (1)		
ADM Spin Edit Scrolling Popup	Control Item	1062	ADMSpinEditPopupStyle	ADMJustify (1)	MenuID =MenuResID (1)		
ADM Text Check Box	Check Box Item Control Item	0	1				
ADM Text Edit	Edit Text Item Control Item	1070	ADMTextEditStyle	ADMJustify (2)			
ADM Text Edit Multi Line	Control Item	1073	0	ADMJustify (1)			
ADM Text Edit Popup	Control Item	1071	ADMTextEditPopupStyle	ADMJustify (1)		MenuID =MenuResID (1)	
ADM Text Edit Scrolling Popup	Control Item	1075	ADMTextEditPopupStyle	ADMJustify (1)		MenuID =MenuResID (1)	

TABLE 1.5 Macintosh ADM Items

ADMItem Type	Mac Dialog Item	Mac CNTL Resource Settings		Additional Fields			
		CDEF Res ID	Variation	Value	Min	Max	Other
ADM Text Push Button	Push Button Item	0	0				
	Control Item	0	4=Default				itemID = 1 is made default automatically
ADM Text Radio Button	Radio Button Item	0	2				
ADM Text Static	Static Text Item	1072	0	ADMJustify (1)			
	Control Item						
ADM Text Static Multi Line	Control Item	1074	0	ADMJustify (1)			
ADM User	Control Item	1080	0				
ADM Custom	Control Item	1090	CNTL Title = "Name Registered Custom Item Type"				
ADM Hierarchical List	Control Item	1011	0				
ADM Progress Bar	Control Item	5	0	IntValue (1)	IntMin (1)	IntMax (1)	
ADM Chasing Arrows	Control Item	7	0				
ADM Dial	Control Item	1045	0	IntValue (1)	IntMin (1)	IntMax (1)	
ADM Tabbed Menu	Deprecated in V2.8 —do not use						

Event Callbacks

Using Event Callbacks

The events received by an ADM object are:

- Init—See [Using Init Functions](#)
- Draw—See [Using Drawer Functions](#)
- Track—a low level event such as from the mouse or keyboard—See [Using Tracker Functions](#)
- Notify—when the object receives a high level event, such as the object was clicked—See [Using Notifier Functions](#)
- Destroy—See [Using Destroy Functions](#)

In general, each ADM object has a default function for each event. If you only need the normal behavior of an item, you can ignore its events and handler functions and rely on the defaults.

If you want some custom behavior for an item, its standard handler functions can be replaced by custom handler functions. The new handler function will likely call the default function and then supplement this behavior. Custom handlers for draw, track, and notify events are called Drawers, Trackers, and Notifiers, respectively. Custom init and destroy functions are implemented using standard C and API functions. ADM Tracker, ADM Drawer, and ADM Notifier functions can also use their related suites. Events are received by all objects in the object container hierarchy. For instance, if the object is an ADM button item, the ADM item receives the event followed by its containing ADM dialog.

Replacing a dialog or item event handler function is done using definitions and functions in the ADM suite for the object type. ADM Entry and ADM List Entry objects can have custom handler functions, but these are set by their parent list. ADM List objects are handled by their parent dialog item.

If the handler can be changed, there will be a **SetEventProc()** function for the handler in the object suite. If the default handler function can be called there will be **DefaultEvent()** function in the suite. For instance, to override the default drawing behavior of an ADM Item object you would use these definitions and functions:

```
typedef void ASAPI (*ADMItemDrawProc)(ADMItemRef inItem,  
    ADMDrawerRef inDrawer);  
  
void ASAPI (*SetDrawProc)(ADMItemRef inItem, ADMItemDrawProc  
    inDrawProc);  
void ASAPI (*DefaultDraw)(ADMItemRef inItem, ADMDrawerRef inDrawer);
```

The new handler function must follow the correct function prototype, which is defined to have enough information to handle the event. For instance, your custom draw item function would receive the item to draw and a drawer reference used to draw the item.

The ADM objects and events that can have custom handlers are listed in [Table 1.6](#).

TABLE 1.6 *ADM Object Customization*

Event/Customization	ADM Dialog	ADM Item	ADM Entry ¹
Init			
Customizable	Y	Y	Y
Can Call Default	Y	Y	N ²
Draw			
Customizable	Y	Y	Y
Can Call Default	Y	Y	Y
Track			
Customizable	Y	Y	Y
Can Call Default	Y	Y	Y
Notify			
Customizable	Y	Y	Y
Can Call Default	Y	Y	Y
Destroy			
Customizable	Y	Y	Y
Can Call Default	Y	Y	N ²

1) Custom ADM entry functions are set by the parent list.

2) ADM entry item create and destroy functions are called from the parent list object, unlike the draw, track and notify functions, which can be called by the item handler.

Using Init Functions

ADM initialization functions for dialogs and items are passed in when the [sADMDialog->Create\(\)](#) function is called. ADM lists don't have a unique init function, but are treated as ADM items. The only ADM object to which you assign a new init routine is an ADM entry, and this is actually assigned to the parent ADM List object. Each time an entry is added to the list, the initialization function is called.

The general format of init functions is given below. When the init function is called for an object, a reference to the newly created object is passed to it:

```
typedef ASErr ASAPI (*ADMOBJECTInitProc)(ADMOBJECTRef inObject);
```


The example below shows two init functions, one for a dialog and another for an entry. The dialog init function actually sets the entry init function and item handler functions:

```
AS_ERR myDialogInit(ADMDialogRef dialog) {
    ADMItemRef initItem;
    ADMListRef myList;

    initItem = sADMDialog->GetItem(myDialog, kOKButton);
    sADMItem->SetNotifyProc(initItem, myOKHandler);

    initItem = sADMDialog->GetItem(myDialog, kList);
    myList = sADMItem->GetList(initItem);
    sADMList->SetInitProc(myList, myListEntryInit);

    initItem = sADMDialog->GetItem(myDialog, kCustomItem);
    sADMItem->SetDrawProc(initItem, mySquareDrawHandler);
    sADMItem->SetTrackProc(initItem, mySquareTrackHandler);
}

AS_ERR myListEntryInit(ADMEEntryRef entry) {
    // init stuff such as setting a color or a pointer
}
```

The dialog init function is passed to the dialog when it is created with the [sADMDialog->Create\(\)](#) or [sADMDialog->Modal\(\)](#) functions:

```
ADMDialogRef myDialog;
myDialog = sADMDialog->Modal(gPlugInRef, "DialogName", kMyDialogID,
    kADMModalDialogStyle, myDialogInit, NULL, 0);
```

The init function is called whenever an entry is created in the list to which the init function was assigned. This function call in this example causes the function **myListEntryInit()** to be called so the entry can be initialized:

```
ADMEEntryRef someEntry = sADMList->InsertEntry(myList, 0);
```

Using Drawer Functions

ADM dialogs, ADM items, and ADM entries can all have custom draw handlers, which draw the object on the screen. Whenever an object needs to be updated, its ADM drawer is called. ADM drawers may enhance the appearance of a standard object or perform all the drawing of an object. The draw function for an entry is actually set for its parent list and affects all the list's entries.

An ADM drawer function is defined as:

```
typedef void ASAPI (*ADMObjectDrawProc)(ADMObjectRef inObject,
    ADMDrawerRef inDrawer);
```

The object reference is for the object to be drawn. The **ADMDrawerRef** is similar to a platform window reference or port and is where drawing commands are performed.

Drawing is done using the ADM Drawer suite, which contains a set of platform independent graphics functions such as [sADMDrawer->SetADMColor\(\)](#) and

[sADMDrawer->DrawLine\(\)](#). The **ADMDrawerRef** passed to the draw function is passed to each of the graphics functions.

This example of an ADM drawer calls the default draw function for the item and then supplements it by drawing a shadow rectangle around it:

```
void mySquareDrawHandler(ADMItemRef item, ADMDrawerRef drawer) {
    ASRect boundsRect;
    sADMItem->DefaultDraw(item, drawer);

    sADMDrawer->GetBoundsRect(drawer, &boundsRect);
    boundsRect.top -= 2;
    boundsRect.bottom += 2;
    boundsRect.left -= 2;
    boundsRect.right += 2;

    sADMDrawer->SetADMColor(drawer, kADMShadowColor)
    sADMDrawer->DrawRect(drawer, &boundsRect)
}
```

This drawer example is assigned to an item in the code example in [Using Init Functions](#), above.

Using Notifier Functions

Notifiers are probably the event you will most frequently override. A notifier is essentially a notification that a high level system event has occurred. Notifier events occur when a user interacts with an ADM object. Two common notifications are when a dialog is resized or when an **OK** button is clicked. The latter is frequently how settings are extracted from an ADM modal dialog before the dialog is disposed of. ADM notifiers are listed and described in [Table 17.1](#).

An ADM notifier function receives a reference to the object being notified and a notifier reference. The notifier reference is to the event that triggered it. The signature for the callback looks like this:

```
typedef void ASAPI (*ADMObjectNotifyProc)(ADMObjectRef inObject,
    ADMNotifierRef inNotifier);
```

It would be used something like this:

```
void myOKHandler(ADMItemRef item, ADMNotifierRef notifier) {
    sADMItem->DefaultNotify(item, notifier);
    getDialogValues();
}
```

The [sADMItem->DefaultNotify\(\)](#) function call is made to provide the item's standard behavior. The dialog values would be extracted with other ADM Item suite functions. The above notify handler is assigned to a button in the code example in [Using Init Functions](#), above.

Dialog items often interact with each other—for instance, a button might restore the default values of other items. Here is an example of a notifier to accomplish this:

```
void mySetDefaultsButtonHandler(ADMItemRef item, ADMNotifierRef
notifier) {
    ADMDialogRef thisDialog;
    ADMItemRef mySlider;

    sADMItem->DefaultNotify(item, notifier);

    thisDialog = sADMItem->GetDialog(item);
    mySlider = sADMDialog->GetItem(thisDialog, kMySliderItem);

    sADMItem->SetIntValue(mySlider, kDefaultSliderValue);
}
```

Notice that the handler function for the item does not need to use global references to the item with which it interacts. Instead it gets its dialog and then uses this reference to obtain the other item.

The notifier reference passed to a notifier function can be used with the ADM Notifier suite to get more information about the reason for the notifier. For instance, there are several types of actions that trigger a notify event.

```
#define kADMUserChangedNotifier      "ADM User Changed Notifier"
#define kADMBoundsChangedNotifier   "ADM Bounds Changed Notifier"
```

The changed notifier type is the most common reason why a notifier is called and simply means that the user has changed something in the dialog. The bounds changed notifier will be received by an object when it has been resized. To determine which event a dialog notifier has received, you would use the [sADMNotifier->IsNotifierType\(\)](#) function:

```
AS_ERR myResizeItemNotifyHandler(ADMItemRef item,
ADMNotifierRef notifier)
{
    sADMItem->DefaultNotify(item, notifier);

    if (sADMNotifier->IsNotifierType(notifier,
kADMBoundsChangedNotifier))
    {
        ADMDialogRef dialog = sADMItem->GetDialog(item);
        handleWindowResize(dialog);
    }
}
```

Using Tracker Functions

A tracker function is used by an ADM object to monitor low level user events, such as mouse movement and keystrokes, while it is the current object. In most cases, a notifier is sufficient, but when this is not enough, ADM dialogs, items, and entries can have trackers. List entry trackers are set by the parent list and affect all its entries.

An ADM event tracking function is defined as:

```
typedef ASBoolean ASAPI (*ADMObjectTrackProc)(ADMObjectRef inObject,
ADMTrackerRef inTracker);
```

The **ADMTrackerRef** is basically an identifier for the current event. The object reference is for the object receiving the event. If the track function returns **true**, its item will receive a notify event when the mouse is released. For trackers on text items and key events, returning **true** means the key was handled. If it returns **false**, a notify event will not be received.

Information about the event is obtained using the ADM Tracker suite functions. The **ADMTrackerRef** argument is passed to a function in the suite and event information is returned.

This example of an ADM tracker function checks for and handles a shift click. A normal click would be handled by the button's notifier function:

```
ASBoolean mySquareTrackHandler(ADMItemRef item, ADMTrackerRef tracker) {
    Boolean shiftKeyDown, notify = true;
    ADMAction thisAction;

    shiftKeyDown = sADMTracker->GetModifiers(tracker) ==
        kADMShiftKeyDownModifier;
    thisAction = sADMTracker->GetAction(tracker);

    if ((action == kADMButtonDownAction) && shiftKeyDown) {
        handleShiftClick();
        sADMTracker->Abort();
        notify = false;
    }
    return notify;
}
```

The tracker function example above is assigned to an item in the code example for the example in [Using Init Functions](#), above.

Using Destroy Functions

A destroy handler function is where you do any necessary clean up for an object about to be deleted from memory. It is triggered by a plug-in calling the **Destroy()** function on the object. A destroy function is passed a reference to the object about to be destroyed and is defined as:

```
typedef void ASAPI (*ADMObjectDestroyProc)(ADMObjectRef inObject);
```

If an init function allocated memory, it should be de-allocated here.

Using Resizable Windows

If a resizable window grows or shrinks, the resizing and relocating of dialog items must be handled. This event is sent to the dialog's resize items notifier function, so

adding your own notify handler and checking that the notify event type is an **kADMBoundsChangedNotifier** event will allow you to handle the resize. See [Using Notifier Functions](#), above, for more information.

Custom Item Types

NOTE: Custom item types are deprecated in ADM V2.8.

A mechanism has been provided through which new ADM item types can be added. These custom item types are then usable by other clients in their ADM dialogs. The provider is responsible for drawing the item for all subscribers and maintaining information needed to do so.

An example of a custom ADM Item can be seen in the Adobe Illustrator application tool palette. The fill and stroke color indicators at the bottom of the tool palette are a custom ADM Item provided by the paint style plug-in.

To add a new ADM item type, use the [sADMDialog->RegisterItemType\(\)](#) at startup.

```
error = sADMDialog->RegisterItemType(gPlugInRef, kMyADMCustomType);
```

When any client creates a dialog with the added custom type, the provider will receive a PICA event to create it in an ADM dialog window. The provider will be called through its main entry point with the following information:

```
caller == kADMCaller
selector == kADMCreateCustomItem
message == ADMCreateCustomItemMessage*
```

The **ADMCreateCustomItemMessage** data structure looks like this:

```
typedef struct
{
    SPMessageData d;
    struct ADMDialog *dialog;
    int itemID;
    ADMItemType itemType;
    ASRect boundsRect;
    ADMItemInitProc initProc;
    ADMUserData data;
    struct ADMItem *item;
}
ADMCreateCustomItemMessage;
```

The provider needs to respond to this event by creating an item within the indicated dialog, in the location specified by the bounds rectangle (*boundsRect*) in the message structure. This is done with the [sADMItem->Create\(\)](#) function:

```
message->item = sADMItem->Create(message->dialog, message->itemID,
    kADMUserType, &message->boundsRect, initProc, userData);
```

The event handler functions for the newly created item *must all be overridden*. The new handlers would handle the drawing and user events for the item. In the above

example an ADM User Type is used. You may be able to use another ADM Item type if you want to use some of its functionality.

If a plug-in provides the custom item, it is responsible for acquiring itself to ensure that it is not unloaded from memory. This is done using the PICA constant, **kSPAccessSuite**. When it no longer is supporting any ADM Items, the providing plug-in can release itself and be unloaded.

When the item is no longer needed (it or its containing dialog is destroyed), the destroy function that you provided will be called. You will need to do any cleanup associated with the item at this point.

Using Custom Items

If a plug-in wants to use a custom item, it is a much simpler process. The plug-in simply includes an item of type **kADMCustomType** in its resource item list. When ADM loads and creates the dialog, it will notice the custom item and tell the providing plug-in to create and maintain it. To indicate a custom item, the resources listed in [Table 1.7](#) and [Table 1.8](#) are used.

TABLE 1.7 Windows Custom ADM Item Resource Information

Windows Class Name	Window Style
<custom item name>	""

TABLE 1.8 Macintosh Custom ADM Item Resource Information

Mac Dialog Item	CDEF Res ID	Control Title
Control Item	1090	<custom item name>

The custom item can also be created at runtime with the [sADMDialog->CreateItem\(\)](#) function:

```
sADMDialog->CreateItem(myDialog, kMyCustomItemID,
    <custom item name>, &myCustomItemRect);
```

The custom item may use standard ADM Item suite functions to initialize it, a function suite made available by the providing plug-in, or a combination of both.

Using Timer Procedures

It is often useful or necessary to use a timer function to provide a time-out or institute a custom reaction to the activities (or non-activities) of the plug-in user. ADM supports this need with several timer functions available in the suites. They all operate in the same fashion.

An ADM timer function takes an item or dialog reference, a duration in milliseconds, and two callback procedures. The first callback procedure, the completion proc, will be called if the duration expires. From the completion proc, you can return **true** and ADM will repeat the timer. The second callback will be called if the timer is aborted before the time duration elapses. An abort mask is passed to the timer create procedure and takes an action mask as defined in `ADMTracker.h`. If one of the actions in the mask occurs before the timer duration is finished, the abort proc is called. The action that caused the timer abort is passed to the callback.

Using the C++ Interfaces

NOTE: The C++ interfaces are deprecated in ADM V2.8. They will no longer be supported.

There are two sets of interfaces for working with ADM, a standard C interface and a C++ interface. The C++ interface puts an object oriented wrapper around the procedural API and you may find them more convenient to use. They eliminate the need to specify and de-reference the suite pointer and the need to pass the ADM object being processed. These wrappers can be found in the `IADM` (Interface to ADM) directory in several Adobe SDKs. The following example shows the same process in both styles:

```
// Using ADM in a procedural manner
ADMItem theItem;
ADMList theList;

theItem = sADMDialog->GetItem(kADMMenuItemID);
theList = sADMItem->GetList(theItem)
sADMList->SetMenuID(theItemsList, gPlugInRef, 16000, "Choices");
sADMItem->Enable(theItem, true);

// Using ADM as objects
IADMList theList = GetItem(kADMMenuItemID).GetList();
theList.SetMenuID(kColorMenuID);
GetItem(kADMMenuItemID).Enable();
```

There are ADM C++ wrapper classes for each of the ADM object event suites. Each of these classes basically repackages a corresponding set of suite functions into an object definition, for instance:

```
class IADMDialog
class IADMItem
class IADMList
class IADMEntry

class IADMNotifier
class IADMDrawer
class IADMTracker
```

The convention used for the definition files is to add an “I” (for “Interface”) at the beginning of the standard ADM suite (e.g. “IADMItem.hpp”).

In addition to the wrapper classes, two base classes are provided to aid in creating C++ based ADM dialogs:

```
class BaseADMDialog
class BaseADMItem
```

These classes provide the basic constructors and destructors for ADM objects and provide a means for overriding the event callbacks. There is no support for custom event functions in the interface wrappers; these are handled in the base classes. You can use these as a foundation for building your own dialogs. The source and header files for these classes are provided in .cpp and .hpp files of the class name.

To use the C++ interfaces you must use these globals for ADM suites:

```
ADMBasicSuite *sADMBasic;
ADMDialogSuite *sADMDialog;
ADMDialogGroupSuite *sADMDialogGroup;
ADMDrawerSuite *sADMDrawer;
ADMEntrySuite *sADMEntry;
ADMHierarchyListSuite *sADMHierarchyList;
ADMIconSuite *sADMIcon;
ADMImageSuite *sADMImage;
ADMItemSuite *sADMItem;
ADMListSuite *sADMList;
ADMListEntrySuite *sADMListEntry;
ADMNotifierSuite *sADMNotifier;
ADMTrackerSuite *sADMTracker;
```

Getting Started With ADM Plug-In Development

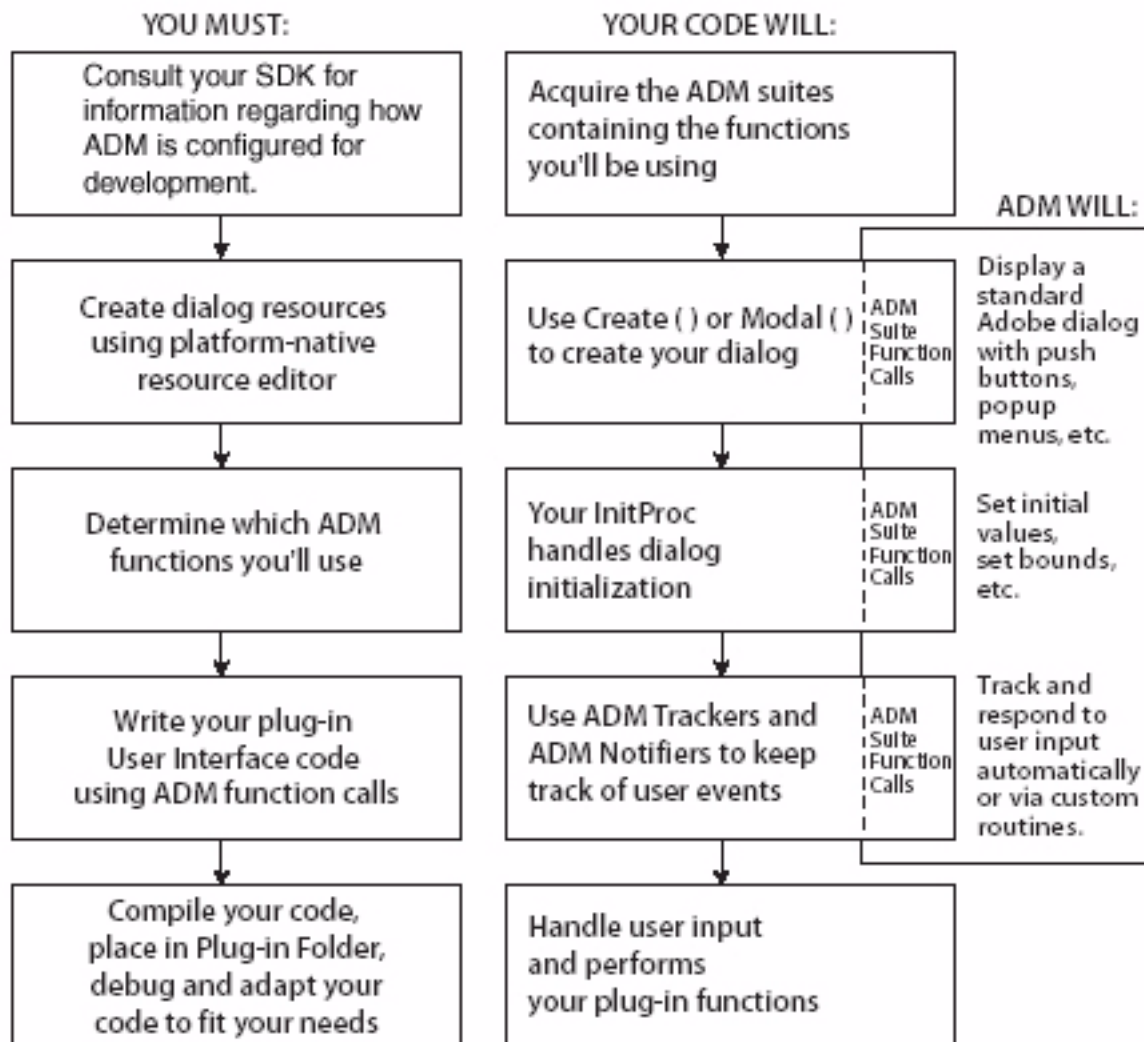
The easiest way to get started quickly with ADM is to examine the sample code found in the various SDKs available for the host applications using ADM (Acrobat, Photoshop, Illustrator, and After Effects) and adapt it to your needs. These can be downloaded from:

<http://partners.adobe.com>

Once you have implemented the basic dialogs, you can refine and add any of the many features supported by ADM.

The General Development Process

The process of using ADM is shown below:



By using ADM, you can greatly reduce the time required to create robust dialogs that conform to the host application appearance. This frees you to focus your energies on your plug-in functionality.

There are some basic requirements to using ADM—for example, for some applications the ADM plug-in module must be placed in the host application's Plug-in Folder in order to be available to the plug-in.

Second, whatever ADM features your plug-in uses must be acquired via PICA suite calls. This will generally be the first thing that your plug-in code will do. Then, when your plug-in calls for an ADM feature, the appropriate ADM functions will already be loaded into memory and be ready to go. Typically, you acquire only the suites you actually need.

NOTE: Some SDKs (for example, the Acrobat SDK) handle the second step for you with special code provided for rapid development.

While ADM eliminates much of the work associated with handling dialogs and is supported on both Windows and Macintosh platforms, the actual visual dialog resources themselves must be created using platform specific resource editors such as Resorcerer and ResEdit on the Macintosh and Microsoft Development Studio on Windows. Once the resources are created, ADM will handle displaying and manipulating them.

In general, your code will contain the following sections pertaining to ADM:

1. Acquire the ADM suites that contain the functions you'll be using in your plug-in.
2. Use `sADMDialog->Create()` or `sADMDialog->Modal()` to create your dialog
3. Use an Init proc to setup your initial values and parameters
4. Use **Tracker()** and **Notify()** functions to keep track of user events such as mouse overs, mouse clicks, text entries, radio button selections, popup menu selections, etc.
5. Use **Destroy()** to release your dialog (if it is a modeless or floating dialog).

You can make your dialogs as complex as you want and add custom graphics, custom event handlers, etc., but ADM provides a rich set of automatic user event handling and you may find that much of what you need is already built-in to ADM.

2

Using ADM with Adobe Acrobat

Introduction

Acrobat 5.0 exposes ADM functionality to plug-in developers (as well as using ADM to implement its own dialogs). This allows developers to use a single code base to implement dialog behavior on both Macintosh and Windows. In previous versions of Acrobat, developers had to write platform-specific code in order to support both platforms.

NOTE: It is still necessary to create platform-specific dialog resources using the appropriate tools on each platform.

The Acrobat SDK provides some header files to help in the use of ADM (mentioned in sections of this chapter). In addition, several samples in the Acrobat SDK illustrate the use of ADM, including: BatchCommand, PlacedImage, rot13, signDoc, VerifyURLs, and transparency.

Using ADM with Acrobat

As described in [Chapter 1, “ADM Overview”](#), ADM contains a series of suites, each of which implements different functionality. Each suite has its own header file that declares its functions, data types, constants, and so on.

Some of the key suites are:

- **ADMBasicSuite:** provides ways to send simple messages to the user, such as alerts or beeps, and also provides basic utilities needed to support the other ADM suites.
- **ADMDialogSuite:** allows you to create and access ADM Dialog objects.
- **ADMItemSuite:** allows you to create and access ADM Item objects.

ADM is implemented by means of an Adobe plug-in architecture called PICA (also known as *SweetPea*). Acrobat plug-in developers need to use the `AcroViewSweetPeaHFT`, which exposes 6 methods, declared in `AVSPCalls.h`:

```
SPBasicSuite* AVSweetPeaGetBasicSuiteP (void);
SPPluginRef AVSweetPeaGetPluginRef (void);
SPErr AVSweetPeaGetResourceAccess (SPPluginRef pluginRef,
    SPPlatformAccessRef *resourceAccess);
SPErr AVSweetPeaSetResourceAccess (SPPluginRef pluginRef,
    SPPlatformAccessRef resourceAccess);
ASBool AVSweetPeaIsADMAvailable (void);
```

Initializing ADM

The Acrobat SDK provides some routines you can call to make your use of ADM easier. They are declared in the header file `ADMUtils.h` (which is automatically included when you include `DUCallbacks.h`) and implemented in `ADMUtils.cpp`. These routines include:

- **InitializeADM:** This routine acquires all the ADM suites (see the next section, “Acquiring and Using ADM Suites” for details) and sets up resource handling (see “Handling Resources” on page 79 for details). Call this routine before doing any ADM-related tasks.
- **ReleaseADM:** This routine releases all the suites that were acquired. Call this routine when you are finished using ADM.

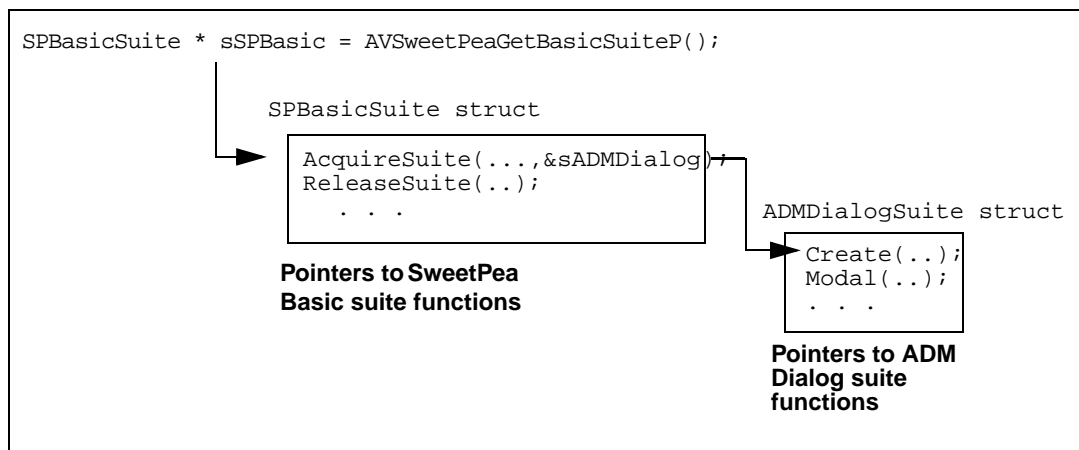
Acquiring and Using ADM Suites

The first step in using an ADM suite is to *acquire* the suite. When the suite is no longer needed, it should be *released*.

The following steps show how suites are acquired. You don’t need to understand the details if you use **InitializeADM**, discussed in the previous section. That routine acquires all the ADM suites and declares pointers to them: **sADMBasic** for the ADM Basic suite, and so on.

The process is illustrated in the following figure:

FIGURE 3 **Acquiring the ADM Dialog Suite Functions**



1. Call the Acrobat method **AVSweetPeaGetBasicSuiteP** to access the SweetPea basic suite.

```
SPBasicSuite* sSPBasic = AVSweetPeaGetBasicSuiteP();
```

This routine returns an **SPBasicSuite***, which is a pointer to a struct (declared in `SPBasic.h`) that contains pointers to the SweetPea functions. **SPBasicSuite** includes the following functions:

```
typedef struct SPBasicSuite {
    SPAPI SPErr (*AcquireSuite)( const char *name, long version,
    const void **suite );
    SPAPI SPErr (*ReleaseSuite)( const char *name, long version );
    ....
} SPBasicSuite;
```

2. Declare pointers to the ADM suites you want to use, as follows:

```
ADMBasicSuite7      *sADMBasic = NULL;
ADMDialogSuite7     *sADMDialog = NULL;
...
```

Each of these items (**ADMBasicSuite7**, etc.) is a structure (defined in the suite's header file) containing a set of pointers to all of the suite's functions. Once you acquire the suite, you use the pointers to access the functions.

NOTE: The name of the suite contains its version number. Acrobat supports the latest version of each suite as indicated in the header files.

3. Call the SweetPea function **AcquireSuite** to acquire the ADM suites you are interested in. For example, to acquire the ADM Dialog suite:

```
sSPBasic->AcquireSuite(kADMDialogSuite, kADMDialogSuiteVersion7,
    (const void **)&sADMDialog);
```

4. Once you have acquired the suites you need, you can access their functions. For example:

```
sADMDialog->Create(...);
```

Note that all functions must be accessed through the suite pointers. In the headers, the ADM suites are structures containing function pointers, as in the following excerpt from `ADMDialog.h`:

```
typedef struct ADMDialogSuite7
{
    // dialog creation
    ADMDialogRef ASAPI (*Create)(SPPluginRef inPluginRef, const
    char* inName, ASInt32 inDialogID, ADMDialogStyle inDialogStyle,
    ADMDialogInitProc inInitProc, ADMUserData inUserData, ASInt32
    inOptions);
    // modal dialogs
    ASInt32 ASAPI (*Modal)(SPPluginRef inPluginRef, const char*
    inName, ASInt32 inDialogID, ADMDialogStyle inDialogStyle,
```

```

    ADMDialogInitProc inInitProc, ADMUserData inUserData, ASInt32
    inOptions);
//...
// dialog destruction
    void ASAPI (*Destroy)(ADMDialogRef inDialog);
    void ASAPI (*SetDestroyProc)(ADMDialogRef inDialog,
    ADMDialogDestroyProc inDestroyProc);

// dialog information accessors
// ...
    ASWindowRef ASAPI (*GetWindowRef)(ADMDialogRef inDialog);
// ... more functions
}
ADMDialogSuite7;

```

5. When you are ready to release the suite, you can do so as follows:

```
sSPBasic->ReleaseSuite(kADMDialogSuite, kADMDialogSuiteVersion7);
```

Using Modal Dialogs

As shown above, you use the ADM Dialog suite function [sADMDialog->Modal\(\)](#) to create modal dialogs and [sADMDialog->Create\(\)](#) to create modeless or tabbed floating dialogs.

When using ADM to create modal dialogs in Acrobat, it is still necessary to use the following routines defined in the Acrobat core API:

```

void AVAppBeginModal (AVWindow window);
void AVAppEndModal (void);

```

AVAppBeginModal prepares Acrobat to display a modal dialog. It takes an **AVWindow** parameter, so it should be preceded by a call to this routine:

```

AVWindow AVWindowNewFromPlatformThing (AVWindowLayer layer,
    ASUns32 flags, AVWindowHandler handler, ASExtension owner, void*
    platformThing);

```

NOTE: The **platformThing** parameter is a platform-specific window reference for the dialog, which can be obtained by calling **GetWindowRef** on the ADM dialog just created:

These Acrobat calls can be made in the initialization procedure for the ADM dialog—the user callback routine specified by the **inInitProc** parameter to [sADMDialog->Modal\(\)](#). For example:

```

ASErr ASAPI MyInitProc (ADMDialogRef dialogRef)
{
    ...
    gAVWindow = AVWindowNewFromPlatformThing (AVWLmodal,
        AVWIN_WANTSKEY, NULL, gExtensionID,
        sADMDialog->GetWindowRef(dialogRef));
    AVAppBeginModal (gAVWindow);
    ...
}

```

```
{
```

Likewise, **AVAppEndModal** must be called to inform Acrobat that a modal dialog is no longer being displayed. This call can be made in the *destroy function*, which is established by the ADM function **SetDestroyProc**, and called when the function **Destroy** is called to remove the dialog.

Also note that on Windows, popping an alert dialog (**sADMBasic->ErrorAlert()**, **sADMBasic->MessageAlert()**, **sADMBasic->QuestionAlert()**, **sADMBasic->YesNoAlert()**) does not always work modally. In these cases, to ensure that they do work modally, use the following code:

```
AVAppBeginModal(NULL);
    //alert call
AVAppEndModal();
```

Handling Resources

Since ADM dialogs require the use of platform-specific dialog resources, there are some issues regarding resource handling. Your plug-in must tell ADM to use its own resource chain (rather than that of Acrobat itself).

NOTE: The SDK routine **InitializeADM** handles this for you. The rest of this section explains the details.

To do this, use the following routines from **AVSPCalls.h**:

```
SPErr AVSweetPeaGetResourceAccess (SPPluginRef pluginRef,
    SPPlatformAccessRef *resourceAccess);
SPErr AVSweetPeaSetResourceAccess (SPPluginRef pluginRef,
    SPPlatformAccessRef resourceAccess);
```

The **pluginRef** parameter refers to the SweetPea plug-in, which is ADM itself (*not* your Acrobat plug-in). It should be obtained by calling this routine:

```
SPPluginRef AVSweetPeaGetPluginRef (void);
```

The **resourceAccess** parameter refers to platform-specific resource information. Before using ADM to display a dialog, call **AVSweetPeaGetResourceAccess** to get the previously used resource value. The following code from the Acrobat SDK file **ADMUtilities.cpp** illustrates this:

```
SPPlatformAccessRef oldResourceHandle = NULL;
sADMPluginRef = AVSweetPeaGetPluginRef();
AVSweetPeaGetResourceAccess (sADMPluginRef, &oldResourceHandle);
```

Then set the value depending on the platform.

```
#ifdef WIN_PLATFORM
    retVal = AVSweetPeaSetResourceAccess(sADMPluginRef,
        (SPPlatformAccessRef)gHINSTANCE;
```

```
#elif MAC_PLATFORM
    SPMacResChain macResInfo;
    sADMResourceContext = new StAcroResourceContext (gResFile);
    //Makes sure plug-in resource file is at top of resource chain.
    macResInfo.fileRefNum = CurResFile();
    retVal = AVSweetPeaSetResourceAccess(sADMPluginRef,
        (SPPlatformAccessRef)&macResInfo;
#endif
```

When finished with a dialog, the resource chain can be set back to Acrobat's:

```
AVSweetPeaSetResourceAccess(sADMPluginRef, oldResourceHandle);
```

Macintosh Issues

Carbon

Carbon is a set of programming interfaces that can be used to build Mac OS X applications that also run on Mac OS 8 and 9 (versions 8.1 and later). It includes a set of header files and a library called `CarbonLib`. These interfaces include most of the functions commonly used by Macintosh developers. By developing to these interfaces and linking with `CarbonLib` (instead of previous libraries such as `InterfaceLib`), a “carbonized” executable is produced which can run under Mac OS X (in native mode) as well as Mac OS 8/9.

The Acrobat 5.0 products are carbonized. In order for a plug-in to use ADM, it must also be carbonized.

NOTE: Plug-ins which do not use ADM will still work on versions other than Mac OS X even if they are not carbonized.

‘carb’ Resources

In order to run in native mode in Mac OS X, plug-ins are required to have an empty ‘carb’ resource of ID 0. The native versions of Acrobat 5.0 and Acrobat Reader 5.0 (when they are released) will ignore plug-ins that do not have a ‘carb’ resource (no errors will be raised).

Development Environment and Documentation

Metrowerks Code Warrior version 6 for Mac OS contains the correct headers and libraries, as well as extensive documentation on making Macintosh applications Carbon-compliant.

Also available from Apple:

<http://developer.apple.com/techpubs/carbon/pdf/CarbonPortingGuide.pdf>

3

Using ADM with Adobe Photoshop

Introduction

Photoshop does not support all of the functionality of ADM. In particular, it supports only modal dialogs and does not support modeless or floating/tabbed dialogs. ADM is primarily used in Photoshop to present dialog boxes for use with utility functions that are available from menu entries.

This chapter explains how the ADM plug-in code works for a specific Photoshop plug-in. To work through this chapter you will need Photoshop 6.0 or later and the Photoshop 6.0 SDK or later (which is available from <http://partners.adobe.com>). You can use these products on either the Macintosh or Windows platforms.

Frame Select Photoshop Plug-in

The example used in this chapter, Frame Select, is a sample automation plug-in available in the `SampleCode` directory of the Photoshop SDK. Automation plug-ins are used to execute Photoshop commands in a programmatic fashion. While designed as a simple sample of an automation plug-in that uses the scriptable actions engine in Photoshop, the Frame Select plug-in is also a good example of how to easily create and manage a Photoshop standard modal dialog using ADM.

Platform-Specific Resources

The first step in writing a plug-in that uses ADM is to create the visual dialog itself using a resource editor. On the Macintosh, a product such as Resorcerer® from Mathemaesthetics, Inc. can be used. On Windows, tools available from within the Visual Studio development environment are commonly used.

The code that is generated in this process is the only platform-specific code you need to write to build an ADM-based plug-in that executes on both Windows and Macintosh platforms. Once the code is written, ADM executes it using the `dialogID` argument to the `sADMDialog->Modal()` function; this is the platform native resource ID that is used in creating the dialog.

For the Frame Select plug-in, the dialog shown in [Figure 3.1](#) was created on the Macintosh platform, while [Figure 3.2](#) was created on the Windows platform.

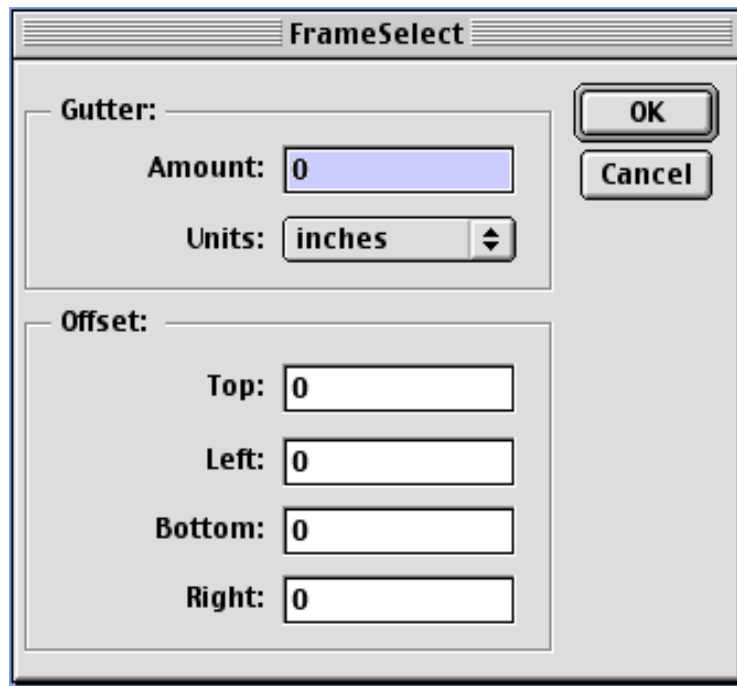


FIGURE 3.1 Dialog Resource Created on a Macintosh

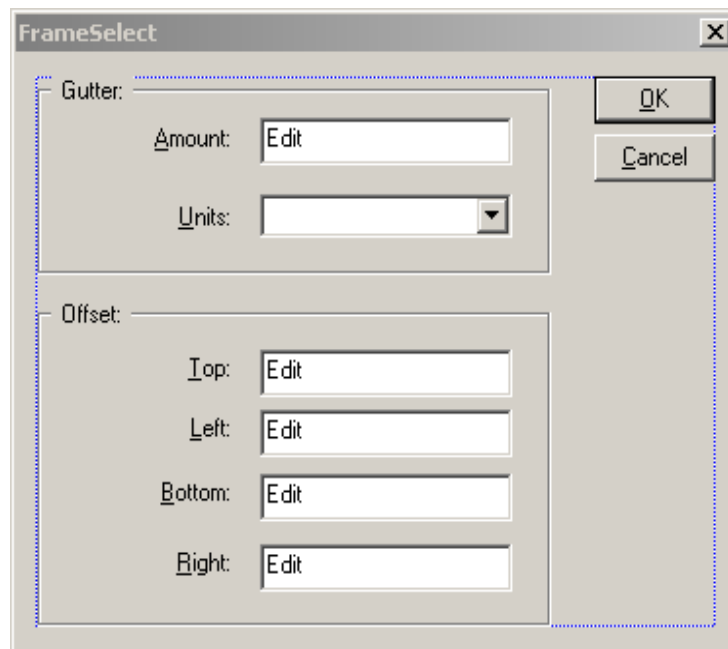


FIGURE 3.2 Dialog resource Created on Windows

The Frame Select plug-in pops these dialogs, filling the various options with default values (shown). It then accepts any user changes or inputs, then, upon dismissal of the dialog via the **OK** button, creates a new frame in the Photoshop document. This frame can then be used to edit the document as required.

The dialogs contain static text items, text edit items, and the **OK** and **Cancel** push buttons. The following is the Windows platform-specific used to generate the dialog show in [Figure 3.2](#). It can be found in the `FrameSelect.rc` file in the Frame Select Visual Studio project:

```
16001 DIALOG DISCARDABLE 0, 0, 256, 191
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "FrameSelect"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON "&OK",1,208,7,41,14
    PUSHBUTTON "&Cancel",2,208,24,41,14
    RTEXT "&Amount:",5,17,21,56,11
    EDITTEXT 6,83,19,86,14,ES_AUTOHSCROLL
    RTEXT "&Units:",7,17,44,56,11
    COMBOBOX 8,83,42,86,17,CBS_DROPDOWNLIST | WS_TABSTOP
    GROUPBOX " Gutter: ",3,7,7,175,59
    RTEXT "&Top:",11,17,89,56,11
    EDITTEXT 12,83,88,86,14,ES_AUTOHSCROLL
    RTEXT "&Left:",13,17,107,56,11
    EDITTEXT 14,83,106,86,14,ES_AUTOHSCROLL
    RTEXT "&Bottom:",15,17,126,56,11
    EDITTEXT 16,83,125,86,14,ES_AUTOHSCROLL
    RTEXT "&Right:",17,17,147,56,11
    EDITTEXT 18,83,146,86,14,ES_AUTOHSCROLL
    GROUPBOX " Offset: ",9,7,87,175,97
END

IDD_DIALOG1 DIALOG DISCARDABLE 0, 0, 186, 95
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Dialog"
FONT 8, "MS Sans Serif"
BEGIN
    DEFPUSHBUTTON "OK",IDOK,129,7,50,14
    PUSHBUTTON "Cancel",IDCANCEL,129,24,50,14
END

#ifdef APSTUDIO_INVOKED
GUIDELINES DESIGNINFO DISCARDABLE
BEGIN
    16001, DIALOG
    BEGIN
        LEFTMARGIN, 7
        RIGHTMARGIN, 249
        TOPMARGIN, 7
        BOTTOMMARGIN, 184
    END
END
```

END

On the Macintosh, the `FrameSelectMacResources.r` file is the resource text file in the Frame Select Code Warrior project that describes the dialog shown in [Figure 3.1](#). It contains the Macintosh **DLOG** and **DITL** dialog box information. Note that the **DLOG** resource describes the entire dialog box information (size, visibility, whether it has a “go away” box, etc.), while the **DITL** resource describes the individual elements within the dialog box (each button, text field, etc.).

```
resource 'DLOG' (kDialogID, plugInName " UI", purgeable) {
    {190, 203, 450, 523},
    movableDialogProc,
    visible,
    noGoAway,
    0x0,
    kDialogID,
    plugInName,
    centerParentWindowScreen
};

resource 'DITL' (kDialogID, plugInName " UI", purgeable) {
    {
        /* array DITLarray: 18 elements */
        /* [1] */
        {8, 252, 28, 312},
        Button {
            enabled,
            "OK"
        },
        /* [2] */
        {34, 252, 56, 312},
        Button {
            enabled,
            "Cancel"
        },
        /* [3] */
        {9, 12, 29, 64},
        StaticText {
            enabled,
            " Gutter:"
        },
        /* [4] */
        {17, 0, 96, 240},
        UserItem {
            enabled
        },
        /* [5] */
        {34, 36, 54, 108},
        StaticText {
            disabled,
            "Amount:"
        },
    },
};
```

```

/* [6] */
{35, 120, 51, 220},
EditText {
    enabled,
    ""
},
/* [7] */
{65, 36, 85, 108},
StaticText {
    disabled,
    "Units:"
},
/* [8] */
{63, 117, 83, 223},
Control {
    enabled,
    16001
},
/* [9] */
{101, 12, 121, 64},
StaticText {
    enabled,
    " Offset:"
},
/* [10] */
{109, 0, 253, 240},
UserItem {
    enabled
},
/* [11] */
{129, 36, 149, 108},
StaticText {
    disabled,
    "Top:"
},
/* [12] */
{130, 120, 146, 220},
EditText {
    enabled,
    ""
},
/* [13] */
{161, 36, 181, 108},
StaticText {
    disabled,
    "Left:"
},
/* [14] */
{162, 120, 178, 220},
EditText {
    enabled,

```

```

        ""
    },
    /* [15] */
    {191, 36, 211, 108},
    StaticText {
        disabled,
        "Bottom:"
    },
    /* [16] */
    {192, 120, 208, 220},
    EditText {
        enabled,
        ""
    },
    /* [17] */
    {221, 36, 241, 108},
    StaticText {
        disabled,
        "Right:"
    },
    /* [18] */
    {222, 120, 238, 220},
    EditText {
        enabled,
        ""
    }
}

};

resource 'CNTL' (kDialogID, "Units", purgeable) {
    {16, 4, 36, 110},
    0,
    visible,
    0,
    kDialogID,
    1016,
    0,
    ""
};

resource 'STR#' (kResetStringID, purgeable)
{
    {
        "Reset"
    }
};

resource 'STR#' (kCancelStringID, purgeable)
{
    {
        "Cancel"
    }
};

```

```

    }
};

resource 'STR#' (kUnitsID, purgeable)
{
    {
        "pixels",
        "inches",
        "cm",
        "points",
        "picas",
        "percent"
    }
};

```

Acquiring the Suites

Before the plug-in code can present the dialog to the user, it must acquire the ADM suites. The ADM suites become available when the ADM plug-in is loaded when Photoshop launches; the process of acquiring the suites is a process of acquiring pointers that can be used to invoke the functions. This is accomplished using the **PIUSuitesAcquire()** function, declared in the `PIUSuites.h` file, which is found in the `SampleCode/Common` directory in the Photoshop SDK. The function has the following signature:

```

SPErr PIUSuitesAcquire(SPBasicSuite* sSPBasic, _AcquireList*
suitesToAcquire, int16 numSuites);

```

The function takes three arguments, as described in [Table 3.1](#).

TABLE 3.1 *Arguments to PIUSuitesAcquire()*

Argument	Description
sSPBasic	A pointer to the SPBasicSuite struct . This struct contains two host member functions: AcquireSuite() and ReleaseSuite() , which are invoked by PIUSuitesAcquire() to acquire the suites.

TABLE 3.1 Arguments to *PIUSuitesAcquire()*

Argument	Description
suitesToAcquire	A pointer to the _AcquireList struct, which is used to pass in the list of suites that need to be acquired, and to get the pointers to the suites. Like PIUSuitesAcquire() , it is also defined in PIUSuites.h : <pre>typedef struct { char* name; int16 version; void** suitePtr; } _AcquireList;</pre>
numSuites	The number of suites to be acquired.

In the Frame Select plug-in code, the call to this function is made in the **FrameSelect.cpp** file. An instance of the **_AcquireList** structure is created (called **MySuites**) and populated with the names and versions of the suites to be acquired, and with **void** placeholder pointers to the suites that will be filled by the call to **PIUSuitesAcquire()**.

The **sSPBasic** pointer is passed to the plug-in when it loads, along with the **gPlugInRef**, which is the unique plug-in ID.

Building, Presenting, and Using the Dialog

The **FrameSelectUI.cpp** file contains the code that is used to initialize and present the dialog, and to set up the notifications and callbacks for the dialog.

FrameSelectUI.cpp includes an enumerated list of the constants of all of the dialog items starting with the **OK** button as item 1 and the **Cancel** button as item 2.

NOTE: Unless otherwise specifically changed by the programmer using a “make default item” function, ADM expects that the default item will be the first item and will act as the **OK** button and that item 2 will be the **Cancel** button.

Below is the enumerated list of dialog items. After assigning a value to the first item, subsequent items will automatically be assigned the next number in order.

```
enum
{
    kDNoUI = -1, // Error.
    kDOK_button = 1, // Must be one.
    kDCancel_button, // Must be two.
    kDGutter_staticText,
    kDGutter_frame,
```



```

kDAmount_staticText,
kDAmount_editText,
kDUnits_staticText,
kDUnits_popUp,
kDOffset_staticText,
kDOffset_frame,
kDTop_staticText,
kDTop_editText,
kDLeft_staticText,
kDLeft_editText,
kDBottom_staticText,
kDBottom_editText,
kDRight_staticText,
kDRight_editText
};

```

The **k** designation indicates a constant. The **D** designation is an Adobe convention used to indicate that these constants are dialog box related.

NOTE: The order follows the numbering of the items in the resource 'DITL' on the Macintosh platform. There is a similar correspondence on the Windows platform.

The dialog is popped in the **DoUI ()** function. Refer to the [sADMDialog->Modal\(\)](#) function documentation for a description of the parameters. One of the parameters that is passed is the initialization function. For this plug-in, the name of the initialization function is **DoUIInit ()**, and follows the calling conventions for the **ADMDialogInitProc** callback as documented in the [sADMDialog->Modal\(\)](#) function. The initialization function initializes each item in the dialog, provides focus to the first text field (**Amount:**) in the dialog, and installs a tracker to change other items as the user enters text into the first text field. Since the code required to initialize the **Cancel** button and the pop-up menu is extensive, separate functions are used to accomplish these tasks.

Operation

The **DoUI ()** function handles interaction with the UI items that make up the dialog. If the user presses the **Cancel** button, it restores the original frame parameter values and exits. Otherwise, the parameters are updated based on any changes the user made while interacting with the dialog.

```

SPErr DoUI ( )
{
    SPPerr error = noErr;

    int item = kDNoUI; // Error value.

    SaveParameters(); // Save our parameters, just in case.
    if (sADMDialog != NULL)
    {

```

```

        item = sADMDialog->Modal
        (
            gPlugInRef,
            "FrameSelection",
            kDialogID,
            kADMModalDialogStyle,
            DoUIInit,
            NULL, /* No user data */
            0//InOptions = 0
        );
    }

    if (item != kDOK_button)
    {
        error = 'STOP';
        RestoreParameters();
    }

    return error;
}

```

If the `sADMDialog` pointer has been set to point to the suite, then the modal dialog is generated using the `sADMDialog->Modal()` function call. If the call is successful, then `item` is assigned a valid value by the `sADMDialog->Modal()` function call. This value is the number of the dialog item that was used to dismiss the dialog, usually the `kDOK_button` or `kDCancel_button` (items 1 and 2). If it is not the `kDOK_button`, then the original frame parameters are restored and the appropriate error is returned.

The `sADMDialog->Modal()` function, as coded here, takes the following parameters.

Parameter	Value	Description
<code>pluginRef</code>	<code>gPlugInRef</code>	Several ADM suites and functions require the plug-in reference of the current plug-in. ADM uses that information to track who owns what dialog. The value assigned to the global <code>gPlugInRef</code> is passed in by the host application via <code>PluginMain()</code> , which is called when the plug-in is first loaded.
<code>name</code>	<code>FrameSelection</code>	The name of the new dialog. In this case we choose FrameSelection .
<code>dialogID</code>	<code>kDialogID</code>	A dialog ID (uiID) is provided. It is the same ID as used in the <code>FrameSelect.r</code> file. This tells ADM what resource to use for the dialog. The cross-platform ADM routines draw the user interface using the platform specific dialog resources discussed earlier in this chapter. In our plug-in, uiID is defined as 16000 . (On Macintosh, see <code>PIUtilities.r</code> for those define statements.)

Parameter	Value	Description
style	kADMModal-DialogStyle	A constant that specifies the style of dialog. In this case (and in almost all Photoshop plug-ins), kADMModalDialogStyle is used. This dialog style supports a window that can be moved around the screen, and that must be dismissed before continuing.
initProc	DoUIInit	The name of the ADMDialogInitProc callback function that initializes the dialog elements using ADM routines. In this case, it is the DoUIInit () function.
data	NULL	Any extra data that may be required. This information can be a pointer, strings, structures, or whatever you may want to associate with your dialog boxes. In this case, there is no user data, so we pass NULL .
inOptions	0	Not used here. See sADMDialog->Modal() .

The first function that is executed when **DoUI ()** is run is **DoUIInit ()**:

```
static ASErr ASAPI DoUIInit(ADMDialogRef dialog)
{
    // SPerr = Sweet Pea / PICA error type;
    // OSErr = Photoshop uses the Operating system type;
    // ASerr = SPerr = "Adobe Systems" error type;
    ASerr error = kSPNoError;

    ADMItemRef item;

    // Set up list and display default item:
    InitCancelButton(dialog);

    // Set up pop-up menu:
    InitUnitsPopUp(dialog);

    // Set up group menu to notify us when changed:
    item = sADMDialog->GetItem(dialog, kDUnits_popUp);
    sADMItem->SetNotifyProc(item, DoUnitsPopUp);

    // Set up Amount edit text:
    item = sADMDialog->GetItem(dialog, kDAmount_editText);
    sADMItem->SetMaxTextLength(item, kMaxTextLength);
    sADMItem->SetUnits(item, kADMNoUnits);
    sADMItem->SetPrecision(item, kPrecision);
    sADMItem->SetMinFloatValue(item, kTopMin);
    sADMItem->SetMaxFloatValue(item, kTopMax);
    sADMItem->SetNotifyProc(item, DoAmountEditText);

    // Install tracker to change other items as text is
    // edited:
    sADMItem->SetTrackProc(item, TrackAmountEditText);
}
```

```

// Highlight edit text as first item:
sADMItem->Activate(item, true);
sADMItem->SelectAll(item);

// Set up Top edit text:
item = sADMDialog->GetItem(dialog, kDTop_editText);
sADMItem->SetMaxTextLength(item, kMaxTextLength);
sADMItem->SetUnits(item, kADMNoUnits);
sADMItem->SetPrecision(item, kPrecision);
sADMItem->SetMinFloatValue(item, kTopMin);
sADMItem->SetMaxFloatValue(item, kTopMax);
sADMItem->SetNotifyProc(item, DoTopEditText);

// Set up Left edit text:
item = sADMDialog->GetItem(dialog, kDLeft_editText);
sADMItem->SetMaxTextLength(item, kMaxTextLength);
sADMItem->SetUnits(item, kADMNoUnits);
sADMItem->SetPrecision(item, kPrecision);
sADMItem->SetMinFloatValue(item, kLeftMin);
sADMItem->SetMaxFloatValue(item, kLeftMax);
sADMItem->SetNotifyProc(item, DoLeftEditText);

// Set up Bottom edit text:
item = sADMDialog->GetItem(dialog, kDBottom_editText);
sADMItem->SetMaxTextLength(item, kMaxTextLength);
sADMItem->SetUnits(item, kADMNoUnits);
sADMItem->SetPrecision(item, kPrecision);
sADMItem->SetMinFloatValue(item, kBottomMin);
sADMItem->SetMaxFloatValue(item, kBottomMax);
sADMItem->SetNotifyProc(item, DoBottomEditText);

// Set up Right edit text:
item = sADMDialog->GetItem(dialog, kDRight_editText);
sADMItem->SetMaxTextLength(item, kMaxTextLength);
sADMItem->SetUnits(item, kADMNoUnits);
sADMItem->SetPrecision(item, kPrecision);
sADMItem->SetMinFloatValue(item, kRightMin);
sADMItem->SetMaxFloatValue(item, kRightMax);
sADMItem->SetNotifyProc(item, DoRightEditText);

UpdateRectValues(dialog);

// Set up justification and style for static text items:
item = sADMDialog->GetItem(dialog, kDAmount_staticText);
sADMItem->SetJustify(item, kADMRightJustify);

item = sADMDialog->GetItem(dialog, kDUnits_staticText);
sADMItem->SetJustify(item, kADMRightJustify);

item = sADMDialog->GetItem(dialog, kDTop_staticText);

```

```

sADMItem->SetJustify(item, kADMRightJustify);

item = sADMDialog->GetItem(dialog, kDLeft_staticText);
sADMItem->SetJustify(item, kADMRightJustify);

item = sADMDialog->GetItem(dialog, kDBottom_staticText);
sADMItem->SetJustify(item, kADMRightJustify);

item = sADMDialog->GetItem(dialog, kDRight_staticText);
sADMItem->SetJustify(item, kADMRightJustify);

return error;
}

```

This function starts by initializing its error condition as no error. Then a variable **item** of type **ADMItemRef** is declared. ADM expects all dialog objects to have their own native type so that it can perform the appropriate type checking. In this case, we are creating a reference to a type **ADMItem**. Other possible ADM types include: **ADMDialog**, **ADMDrawer**, **ADMList**, **ADMEEntry**, **ADMNotifier**, **ADMTracker**, **ADMIcon**, **ADMImage**, **ADMUserData**, **ADMTimer**, **ADMActionMask** and **ADMChar**. These are defined elsewhere in this documentation and listed in the `ADMTypes.h` file.

Next this function sets up the **Cancel** button to notify our UI routine when it is clicked by the user. Since we want to be able to support the Photoshop convention of using the **Cancel** button as a “reset to default parameters” function (when the user holds down the **Option/Alt** key when pressing **Cancel**), we must be notified when the **Cancel** button is pressed. The call to **InitCancelButton(dialog)** sets up these parameters. **InitCancelButton()** is discussed after **DoUIInit()**.

Next the group pop-up menu is created. First the list is cleared, then a new list created. Then a notification is set for when the user changes units.

After that the edit text dialog boxes are created for **Amount:**, **Top:**, **Left:**, **Bottom:**, and **Right:**, focus is provided to the first text field (**Amount:**), and a tracker is installed to simultaneously update the other text fields as the user enters text into the **Amount:** text field.

The various options (maximum text length, units, precision, minimum value, and maximum value) are handled using ADM function calls as shown below for the **Amount:** dialog. This is accomplished using the `sADMItem->SetMaxTextLength()`, `sADMItem->SetUnits()`, `sADMItem->SetPrecision()`, `sADMItem->SetMinFloatValue()`, and `sADMItem->SetMaxFloatValue()` functions.

```

// Set up Amount edit text:
item = sADMDialog->GetItem(dialog, kDAmount_editText);
sADMItem->SetMaxTextLength(item, kMaxTextLength);
sADMItem->SetUnits(item, kADMNoUnits);
sADMItem->SetPrecision(item, kPrecision);
sADMItem->SetMinFloatValue(item, kTopMin);
sADMItem->SetMaxFloatValue(item, kTopMax);
sADMItem->SetNotifyProc(item, DoAmountEditText);

```

Finally `sADMItem->SetNotifyProc()` is used to set the callback that is invoked when the user types text into the text box. The callback in this example is named `DoAmountEditText`, and follows the format defined for callbacks of this type: `ADMItemNotifyProc`. The format for `ADMItemNotifyProc` is defined in the documentation for `sADMItem->SetNotifyProc()`.

Notifiers are important ADM functions that are further explained in the later chapters in this documentation. ADM reports on a number of different user actions and provides this information via notify routines. ADM Notifiers track a number of user events: User Changed, Entry Text Changed, Close Hit Notifier, Collapse Notifier, Expand Notifier, Bound Changed, Hide Window Modifier, etc. `ADMNotifier.h` defines all of the different ADM notifiers that are available. The User Changed Notifier is most helpful since it notifies our routines that the user has changed something in our dialog boxes.

The `InitCancelButton()` function is used initialize the cancel button to receive notification when the user presses the **Cancel** button. In this plug-in we want to support the Photoshop convention of allowing dual use of the **Cancel** button. When the user presses the **Option** key while clicking **Cancel**, the function performed is “reset to default values” and the **Cancel** button displays the **Reset** label instead of the **Cancel** label. The dialog routines must be notified when this condition occurs. ADM makes it possible to track this user activity and to change the button labels on the fly.

`InitCancelButton()`, shown below, takes a reference to the current dialog box and returns any error that occurs while initializing the **Cancel** button. The routine assigns a notifier proc for the **Cancel** button to trap the **Option/Alt** key and mouse-down notifiers and call routines when these events happen.

```
static SPError InitCancelButton(ADMDialogRef dialog)
{
    SPError error = kSPNoError;

    if (dialog != NULL)
    {
        // Set up "Cancel" button to notify us when its been clicked:
        ADMItemRef item = sADMDialog->GetItem(dialog,
            kDCancel_button);
        if (item != NULL)
        {
            sADMItem->SetNotifyProc(item, DoCancel);

            // Set up name of Cancel button. Since we have to have
            // resources around for "Reset" and "Cancel", we might
            // as well check them to load the right value:
            if (gCancelButtonIsReset)
                SetTextToReset(item);
            else
                SetTextToCancel(item);

            // Set up mask for tracker function:
            ADMActionMask mask = sADMItem->GetMask(item);
```

```

sADMItem->SetMask
(
    item,
    mask |
    kADModKeyDownMask |
    kADModKeyUpMask |
    kADLeaveMask |
    kADEnterMask |
    kADButtonUpMask
);

// Install tracker for it to be "Reset" when it needs
to:
sADMItem->SetTrackProc(item, DoReset);
}
else
{
    error = kSPBadParameterError;
}
}
else
{
    error = kSPBadParameterError;
}

return error;

}

```

The routine starts with a no error assignment. It then checks to ensure that the dialog is not **NULL**; then it uses the ADM Dialog suite:

```
ADMItemRef item = sADMDialog->GetItem(dialog, kDCancel_button);
```

to obtain a reference to the **Cancel** button of the current dialog. We then set up the notifier to call our routine when the button is pressed:

```
sADMItem->SetNotifyProc(item, DoCancel);
```

Since we can have two conditions for our **Cancel** button (**Cancel** and **Reset**) we set the text to be displayed on the button using the **SetTextToCancel()** or **SetTextToReset()** functions depending upon which condition is true as specified by the state of the boolean variable **gCancelButtonIsReset**.

The **SetTextToCancel()** function, shown below, gets the appropriate **Cancel** text and then uses the [sADMItem->SetText\(\)](#) function to set the text label for the cancel button on the fly.

```

static void SetTextToCancel
(
    /* IN */ ADMItemRef item // Item ID for Cancel button.
)
{
    if (item != NULL)

```

```

    {
        char text[256];
        sADMBasic->GetIndexString(gPlugInRef, kCancelStringID, 1,
            text, 256);
        sADMItem->SetText(item, text);
    } // item null
}

```

Likewise, the **SetTextToReset ()** function, shown below, does the same thing for the **Reset** label:

```

static void SetTextToReset
(
    /* IN */      ADMItemRef  item    // Item ID for Cancel button.
)
{
    if (item != NULL)
    {
        char text[256];
        sADMBasic->GetIndexString(gPlugInRef, kResetStringID, 1,
            text, 256);
        sADMItem->SetText(item, text);
    } // item null
}

```

Finally, we must set the mask to be able to check whether the **Option/Alt** key was held down when the Cancel button was pressed. This mask information is set up using [sADMItem->SetMask\(\)](#).

You set up a mask to tell ADM what events report to your tracker. This is done by declaring a mask, getting the current mask and then setting a new mask to track the additional events you want:

```

ADMActionMask mask = sADMItem->GetMask(item);
sADMItem->SetMask
(
    item,
    mask |
    kADModKeyDownMask |
    kADModKeyUpMask |
    kADMLLeaveMask |
    kADMEnterMask |
    kADMButtonUpMask
);

```

To track the **Option/Alt** key event, we use a tracker callback function. ADM must be set up to call our track routine when an event occurs:

```

sADMItem->SetTrackProc(item, DoReset);

```

We are choosing to track whether the **Mod** key (the **Option** key on the Macintosh and **Alt** key on the Windows platform) is up or down, and whether the mouse leaves or enters the **Cancel** button, and when the user actually clicks. This lets us control how

the **Cancel** button turns into the **Reset** button (when the **Option** key is pressed and the mouse is over the **Cancel** button) and when the reset is actually performed (upon click with the **Option/Alt** key pressed).

This routine is called when the cancel button is pressed. It is a notifier-receiver:

```
static void ASAPI DoCancel
(
    ADMItemRef    item,
    ADMNotifierRef notifier
)
{
    if (sADMNotify->IsNotifierType(notifier, kADMUserChangedNotifier))
    { // Correct notifier. Do this:
        if (gCancelButtonIsReset)
        { // Must be reset!
            RestoreParameters(); // Resets.
            ADMDialogRef dialog = sADMItem->GetDialog(item);
            DoUIInit(dialog);
        }
        else
        {
            sADMItem->DefaultNotify(item, notifier);
        }
    }
    else
    {
        sADMItem->DefaultNotify(item, notifier);
    }
}
```

The statement: `if (sADMNotify->IsNotifierType(notifier, kADMUserChangedNotifier))` determines what type of user input has been performed. This is possible because ADM tracks a number of different user actions and provides this information. `ADMNotifier.h` defines all of the different ADM notifiers available. In our case, `DoCancel()` obtains the type of notifier and tests it to see if the **Cancel** button is **Reset** (using the `gCancelButtonIsReset` boolean variable defined at the top of the `FrameSelectUI.cpp` file). If it is, then the previous parameters will be restored, and `DoUIInit()` is called to stuff the default parameters into their edit text fields and reset check and radio button groups.

The `Do***EditText()` callbacks are called when the text is changed so that the value can be acquired and put in a global variable to change the frame. The callbacks were set in the `DoUIInit()` initialization function.

To acquire the new values in the dialog boxes, we use the ADM functions to get the value of the changed dialog box. For example, in the `ConvertEditTextNumber()` function:

```
double value = (double)sADMItem->GetFloatValue(item);
```

This is a good example of the way that ADM simplifies the handling of our dialog boxes. In native code for the Mac (or PC) to obtain the value of the user input we would have to acquire the string data, and then convert to integer values, etc.

Finally, we use **SaveParameters()** and **RestoreParameters()** to save and restore our parameters.

4

Using ADM with Adobe Illustrator

A Modeless Dialog Example Using Illustrator

NOTE: The following example was drawn from code that shipped with the Illustrator 7 SDK. However, most of the code still applies to the Illustrator 8, 9, and 10 SDKs. Some of the ADM APIs used in these SDKs pre-date the information in this documentation. However, in virtually all cases, updates of the APIs simply consisted of adding parameters to existing parameters, and the older parameters still work the same. Please see the header files in your SDK to note differences.

The Illustrator SDK, available from <http://partners.adobe.com>, provides an ADM Non-Modal Dialog plug-in (called `ADMNonModalDialogProject`) that demonstrates the process of creating a modeless (or floating) dialog. While it does not actually do anything but put up a floating dialog, it shows all of the ADM function calls required to manage popup menus and lists.

Unlike Photoshop, Illustrator supports all ADM functionality including both modal and modeless dialogs.

The first thing is to create the visual dialog resource. You may want to start with the ADM Non-Modal Dialog plug-in resource and adjust its parameters to fit your needs.

On the Macintosh, you will use a visual editor such as Resorcerer® from Mathemaesthetics, Inc to create your resources. On Windows, you would use the Microsoft Development Studio.

The ADM non-modal dialog created is shown in [Figure 4.1](#).

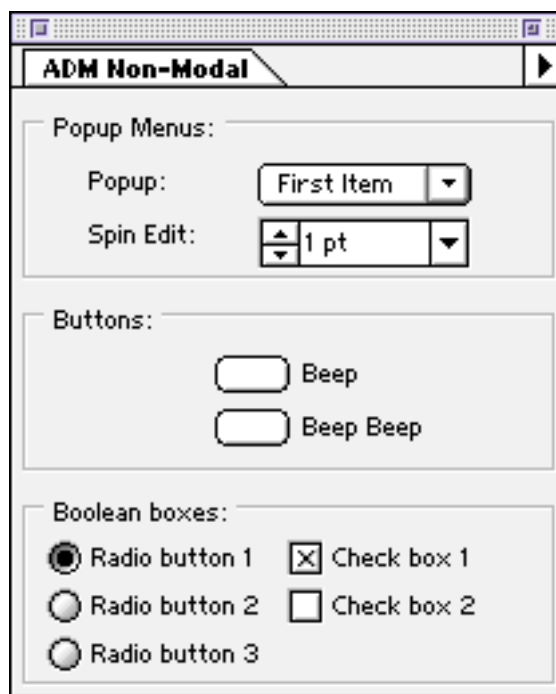


FIGURE 4.1 An ADM Non-modal Dialog

This dialog is the floating tabbed palette type. Note that if you have multiple tabbed dialogs, they can be docked together into a single docked palette—following the conventions in Illustrator, Photoshop, and After Effects. This behavior is handled by ADM and requires little support from your ADM dialog.

Our dialog is created using the `sADMDialog->Create()` function as shown in the code below.

```
ADMDialogRef ASAPI (*Create)(SPPluginRef pluginRef,
    char *name, int dialogID, ADMDialogStyle style,
    ADMDialogInitProc initProc, ADMUserData data);
```

where `message->d.self` obtains the `pluginRef` needed for the function and **DlgInit** is our initialization procedure.

```
// Create the Non-modal dialog. This does not necessarily show the
// dialog on
// the screen. If the dialog was hidden at last shutdown, it will not be
// shown
// until sADMDialog->Show() is called.
// Note: the Init proc, DlgInit, will be called immediately following
// sADMDialog->Create()

AIErr createADMDialog(AINotifierMessage *message) {
    AIErr error = kNoErr;

    g->nonModalDialog = sADMDialog->Create(message->d.self,
        "ADMNonModalDialog", kADMNonModalDialogID,
        kADMTabbedFloatingDialogStyle, DlgInit, nil);

    if (error)
        goto error;
error:
    return error;
}
```

Note that all of the ADM suites are included in the project in the `common.h` file as follows:

```
// ADM Suites
#include "ADMBasic.h"
#include "ADMDialog.h"
#include "ADMItem.h"
#include "ADMIcon.h"
#include "ADMList.h"
#include "ADMDialogGroup.h"
#include "ADMNotifier.h"
#include "ADMEEntry.h"
#include "ADMTracker.h"
```

In our ADM Non-Modal Dialog are various popup menus, push buttons, radio buttons and check boxes.

The constants for these items are defined in the `admHandler.h` file that is included in the `admHandler.c` code. These constants are:

```
#define kADMNonModalDialogID16000

#define kpopupItem5
#define kpopupMenuID 800

#define kbeepItem 3
#define kbeepBeepItem 4

#define kspinEditPopupItem 6
#define kspinEditPopuMenuID 900

#define kradioButton1Item 7
#define kradioButton2Item 8
#define kradioButton3Item 9

#define kcheckBox1Item 10
#define kcheckBox2Item 11
```

Also in the `admHandler.h` file are the prototypes for all of the functions that will be covered in the following discussion as well as some definitions for layer minimum and maximum width and height.

In `admHandler.c`, the initialization procedure, **DlgInit** starts with the declaration of our various ADM variables, **ADMItemRef**, **ADMListRef**, **ADMEEntryRef**, etc.

```
AS_ERR ASAPI DlgInit(ADMDialogRef dlg)
{
    PUSH_GLOBALS
    AS_ERR                                fxErr = kNoErr;
    AIAppContextHandle                    AppContext;
    ADMItemRef                            menuItemRef, popupItemRef,
    beepButtonItemRef,                    beepBeepButtonItemRef,

    spinEditPopupItemRef;
    ADMListRef                            layerListRef = 0;
    ADMListRef                            menuListRef, popupListRef,
    spinEditPopupListRef;
    ADMEEntryRef                          entry;
    SPPluginRef                           pluginRef;
    char                                  tipString[64], groupName[64];

    ASPoint location, size;
    ASRect                                rect, dimensions, boundsRect;
    long                                  positionCode;
    tipString[0] = 0;

    // Set up the application context, so that suite calls will work
    pluginRef = sADMDialog->GetPluginRef(dlg);
    sAIAppContext->PushAppContext(pluginRef, &AppContext);
```

```
// Acquire yourself to stay in memory
sSPAccess->AcquirePlugin(pluginRef, &g->accessRef);

// Acquire suites to stay in memory
acquireSuites(g->basic);
```

Five items are declared as **ADMItemRef**, and these correspond to the five types of items supported in this dialog: a **kADMPopupListType** menu item (labeled **Popup:**), a **kADMPopupMenuType** menu (the right arrow on the right side of the palette), a **kADMSpinEditPopupType** menu item (labeled **Spin Edit**), and two **kADMTextPushButtonType** items (labeled **Beep** and **Beep Beep**).

In addition, the necessary list references are declared in the line:

```
ADMListRef menuListRef, popupListRef, spinEditPopupListRef;
```

The first popup menu is a **kADMPopupListType** menu. It is shown in active position in [Figure 4.2](#).

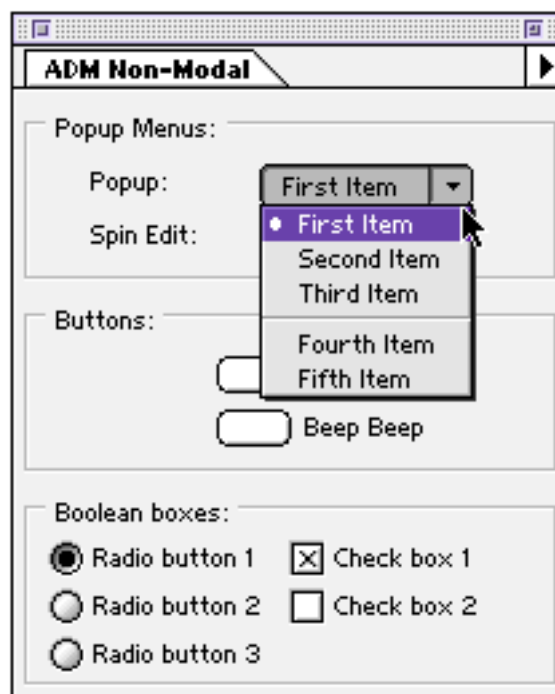


FIGURE 4.2 An ADM Popup List

Below the **PopupList** menu is a spin edit text edit item, shown activated in [Figure 4.3](#).

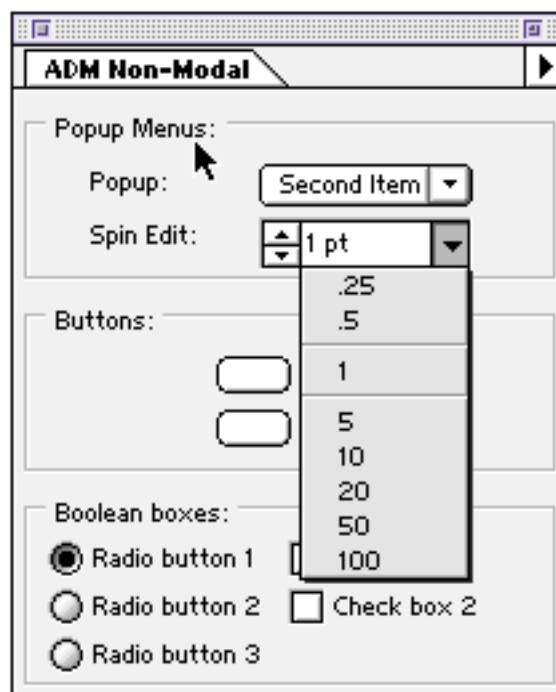


FIGURE 4.3 An ADM Spin Edit Text Item Popup

Also note the right arrow button at the top of the palette. This indicates a `kADMPopupMenuType` menu as shown in [Figure 4.4](#).

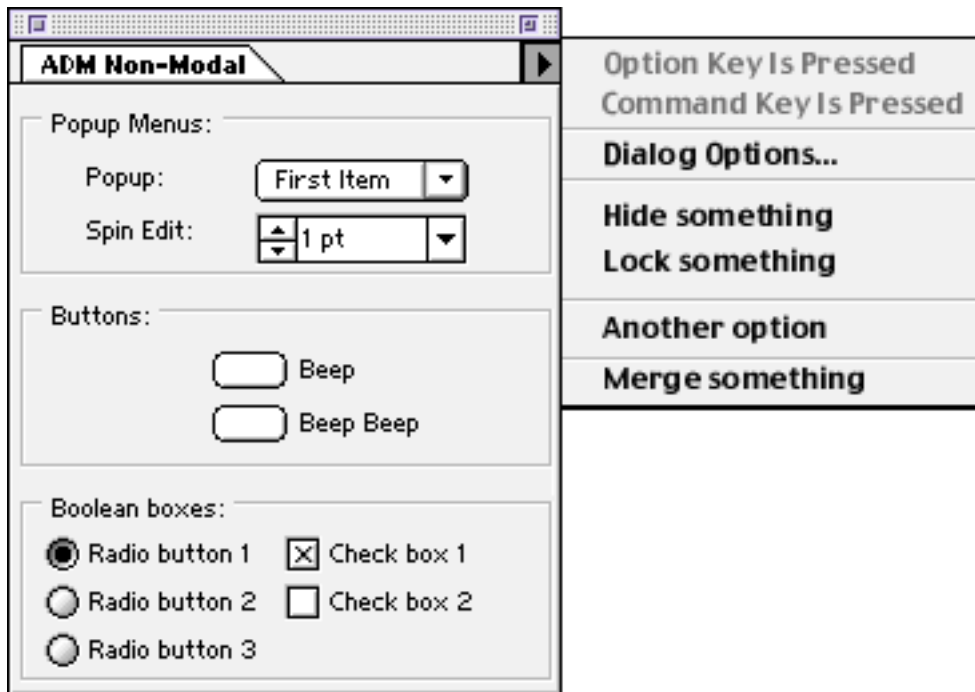


FIGURE 4.4 An ADM Flyout Popup Menu

Note that the first two items are greyed out. They show the ability of ADM to support Mod key effects. In this case, they become active only when in the first case, the **Option** key is being held down when the right arrow menu button is clicked, and in the second case, when the Command key is being pressed when the menu arrow is clicked.

The following code sets up this `kADMPopupMenuType` dialog item:

```
// Setup popup menu on dialog
menuItemRef = sADMDialog->GetItem(dlg, kADMMenuItemID);
sADMItem->SetNotifyProc(menuItemRef, dialogPopupMenuProc);
if (menuItemRef)
{
    menuListRef = sADMItem->GetList(menuItemRef);
    if (menuListRef)
    {
        // 700 is the MENU resource of the list
        sADMList->SetMenuID(menuListRef, 700);
        // CheckPastePref(menuListRef);
    }
    sADMItem->SetTrackProc(menuItemRef, dialogPopupMenuTrackProc);
    // catch mouse down to do setup based on modifier keys
}
```

Once the dialog has been created and all of the buttons and popup menu values established, we want to track the user's activity. We will do this by using ADM Notify and ADM Tracker functions. Note that in the above code, we use

```
menuItemRef = sADMDialog->GetItem(dlg, kADMMenuItemID);
sADMItem->SetNotifyProc(menuItemRef, dialogPopupMenuProc);
```

to set our Notify procedure as **dialogPopupMenuProc**. This will be covered later in this chapter. We also use:

```
sADMItem->SetTrackProc(menuItemRef, dialogPopupMenuTrackProc);
```

to track user activity on this dialog item.

The following code creates the popup list (Popup Menu) and initializes the popup list:

```
// create popup list
popupItemRef = sADMDialog->GetItem(dlg, kpopupItem);
sADMItem->SetNotifyProc(popupItemRef, PopupProc);
popupListRef = sADMItem->GetList(popupItemRef);
sADMList->SetMenuID(popupListRef, kpopupMenuID);

// initialize popup list
entry = sADMList->GetEntry(popupListRef, 1);
// the number you pass is 1 based not 0 based
sADMEEntry->Select(entry, true);
```

This code creates and initializes the spin edit popup list:

```
// create SpinEditPopup list
spinEditPopupItemRef = sADMDialog->GetItem(dlg, kspinEditPopupItem);
sADMItem->SetNotifyProc(spinEditPopupItemRef, spinEditPopupProc);
spinEditPopupListRef = sADMItem->GetList(spinEditPopupItemRef);
sADMList->SetMenuID(spinEditPopupListRef, kspinEditPopuMenuID);

// initialize SpinEditPopup list
entry = sADMList->GetEntry(spinEditPopupListRef, 4);
sADMItem->Invalidate(spinEditPopupItemRef);
sADMEEntry->Select(entry, true);
```

Notice that we are using **PopupProc** and **spinEditPopupProc** as our notify procedures.

At the bottom of the palette is an array of radio buttons and check boxes. ADM provides many useful automatic functions, including the ability group radio buttons together so that within a group, selecting one button automatically deselects the others. ADM considers any consecutively numbered radio button items as a group.

The radio and checkbox buttons are initialized with the following code:

```
// all 3 radio button items will have the same notifier proc.
sADMItem->SetNotifyProc(sADMDialog->GetItem(dlg, kradioButton1Item),
    radioButtonProc);
sADMItem->SetNotifyProc(sADMDialog->GetItem(dlg, kradioButton2Item),
    radioButtonProc);
```

```

sADMItem->SetNotifyProc(sADMDialog->GetItem(dlg, kradioButton3Item),
    radioButtonProc);

// initialize radio button group (items 7-9)
// note: radio buttons with consecutive item numbers will automatically
//       be grouped by ADM
sADMItem->SetBooleanValue(sADMDialog->GetItem(dlg, kradioButton1Item),
    true);
sADMItem->SetBooleanValue(sADMDialog->GetItem(dlg, kradioButton2Item),
    false);
sADMItem->SetBooleanValue(sADMDialog->GetItem(dlg, kradioButton3Item),
    false);

/** Checkbox Items **/

// each check box item will have its own notifier proc.
sADMItem->SetNotifyProc(sADMDialog->GetItem(dlg, kcheckBox1Item),
    checkBox1Proc);
sADMItem->SetNotifyProc(sADMDialog->GetItem(dlg, kcheckBox2Item),
    checkBox2Proc);

// initialize checkboxes
sADMItem->SetBooleanValue(sADMDialog->GetItem(dlg, kcheckBox1Item),
    true);
sADMItem->SetBooleanValue(sADMDialog->GetItem(dlg, kcheckBox2Item),
    false);

```

Finally, we need to setup the button items (**Beep** and **Beep Beep**).

```

// Attach the callbacks for the beep buttons
beepButtonItemRef = sADMDialog->GetItem(dlg, kbeepItem);
if (beepButtonItemRef)
{
    sADMItem->SetNotifyProc(beepButtonItemRef, beepButtonUp);
    // You could also do this stuff:
    // sADMItem->SetItemStyle(buttonItemRef,
    //     kADMBlackRectPictureButtonStyle);
    // sADMItem->SetTrackProc(buttonItemRef, ButtonTrackProc);
    // sADMItem->SetPictureID(buttonItemRef, kADMDeleteEntryPictureID);
    // sADMItem->SetSelectedPictureID(buttonItemRef,
    //     kADMDeleteEntryPictureID);
    // sADMItem->SetDisabledPictureID(buttonItemRef,
    //     kADMDeleteEntryDisabledPictureID);
    // sADMItem->SetCursorID(buttonItemRef, NULL, kADMFingerCursorID);
    // fxErr = sADMBasic->GetIndexString(
    //     pluginRef, kTooltipStrings, trashButtonTipIndex, tipString, 62);
    // if (fxErr)
    //     fxErr = kNoErr; // don't let lack of a tool tip stop the show
    // else if (tipString[0])
    //     sADMItem->SetTipString(buttonItemRef, tipString);
}

```

```

beepBeepButtonItemRef = sADMDialog->GetItem(dlg, kbeepBeepItem);
if (beepBeepButtonItemRef)
{
    sADMItem->SetNotifyProc(beepBeepButtonItemRef, beepBeepButtonUp);
}

```

Unlike a modal dialog, a modeless or floating palette dialog “floats” above the host application and thus can be moved around the screen. Therefore there is some additional overhead to position the dialog.

In addition, palette type dialogs can be combined or docked together into a single palette. For this arrangement, you must be concerned with the **positionCode** value. This sets the dialog's position within a docked/tabbed group.

```

// Get the last known Docking group and Docking code out of the Prefs
file
//sASLib->strcpy(groupName, kLayersPaletteDockGroup);
sAIPreference->GetStringPreference("ADMNonModalDialog",
    "kADM_DPDockGroupStr", groupName);
//positionCode = kLayersPaletteDockCode;
sAIPreference->GetIntegerPreference("ADMNonModalDialog",
    "kADM_DPDockCodeStr", &positionCode);

// Pick a default location in case it has never come up before on this
machine
sADMDialog->GetBoundsRect(dlg, &boundsRect);
sADMBasic->GetPaletteLayoutBounds(&dimensions);
location.h = dimensions.right - (boundsRect.right - boundsRect.left);
location.v = dimensions.bottom - (boundsRect.bottom - boundsRect.top);

// Get the last known location out of the Prefs file
sAIPreference->GetPointPreference("ADMNonModalDialog",
    "kADM_DPLocationStr", &location);

size.h = 208; // minimum width (which governs the inner client rect) + 2
//size.v = layerMinHeight;
size.v = 258;

#ifdef WIN_ENV    // different rules about whether the borders and tabs
                // are in the dlg rect
    size.v += 6;
    location.v -= 6;
    size.h += 4;
#endif
// Get the last known size out of the Prefs file
sAIPreference->GetPointPreference("ADMNonModalDialog",
    "kADM_DPSizeStr", &size);
rect.left = location.h;
rect.right = location.h + size.h;
rect.top = location.v;
rect.bottom = location.v + size.v;

```

```
// restore the size and location of the dialog
sADMDialog->SetBoundsRect(dlg, &rect);
// restore the position code of the dialog
sADMDialogGroup->SetDialogGroupInfo(dlg, groupName, positionCode);

// Initialize the palette internals
//result = LayersDlgInit(dlg);

// Clean up the application context and return
sAIAppContext->PopAppContext(AppContext);
POP_GLOBALS

return fxErr;
}
```

Since we need to be notified when the user interacts with the various dialog elements, we use notification procedures such as **dialogPopupMenuProc** to handle any activity.

```
static void ASAPI dialogPopupMenuProc(ADMItemRef item,
ADMNotifierRef notifier) {
    PUSH_GLOBALS

    int selection;
    // dispatch the notifier type
    if (sADMNotifier->IsNotifierType(notifier,
        kADMUserChangedNotifier) )
        selection = GetPopupSelection(item);
    POP_GLOBALS
}
```

This routine is passed the **ADMItemRef** and the notifier. Once the notifier is checked to make sure that we have the right one (i.e., **kADMUserChangedNotifier**) this means we get the selection by calling **GetPopupSelection()**, as shown below:

```
static int GetPopupSelection(ADMItemRef item) {

    ADMListRef listRef;
    ADMEEntryRef activeEntry;
    int selection;

    listRef = sADMItem->GetList(item);
    // get the current active entry
    activeEntry = sADMList->GetActiveEntry(listRef);
    // get the index (0 based) of the entry
    selection = sADMEEntry->GetIndex(activeEntry);

    return selection;

}
```

As noted above, when the user select the right arrow menu button shown again below, we can check to see if a modifier key is pressed. If so, either one or the other of the

first two items will become selected. Otherwise, they remain greyed out and deselected. As shown below, the **Option** key was pressed while the right arrow was pressed.

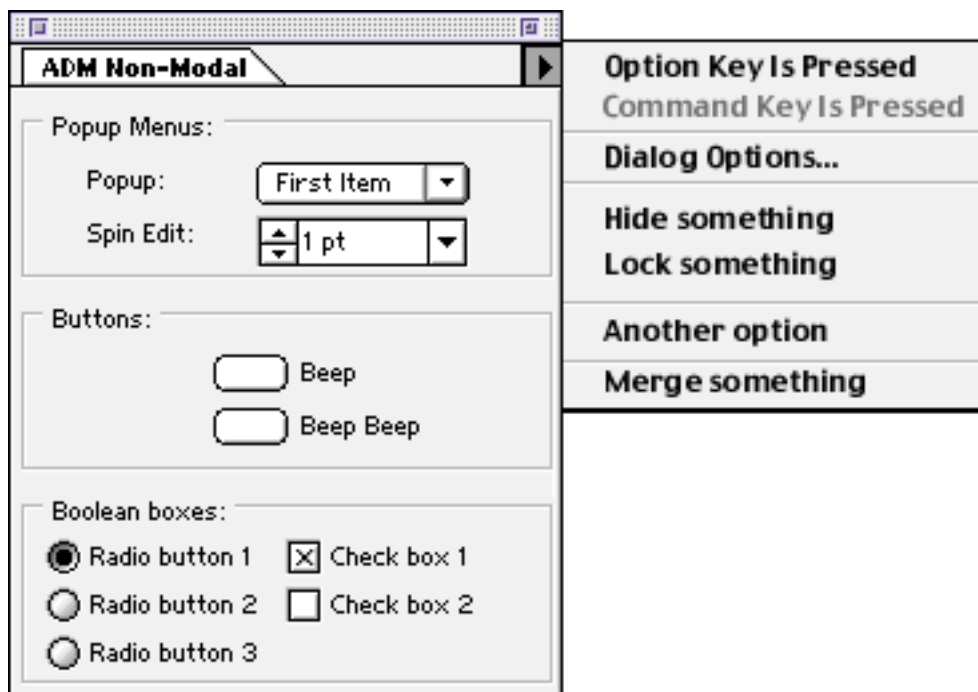


FIGURE 4.5 Flyout Popup Menu with Option Key Pressed

This is handled with the `dialogPopupMenuTrackProc()` procedure:

```
/* dialogPopupMenuTrackProc(), this is called at mouse down on the
"triangle" popup menu. This is your opportunity to change the status of
menu items. In this example, the modifier keys enable menu items 1 and 2
*/
```

```
static ASBoolean ASAPI dialogPopupMenuTrackProc(ADMItemRef item,
ADMTrackerRef tracker)
{
    PUSH_GLOBALS
    ADMAction    action;
    ASBoolean    doNotify;

    action = sADMTracker->GetAction(tracker);

    // capture mouse down event
    if (action == kADMBUTTONDOWNACTION)
    {
        SPPluginRef    pluginRef;
        ADMListRef     menuListRef;
```

```

        AIBoolean          commandOptionDown, commandControlDown;

        // checks if the (mac)OPTION (win)ALT modifier is pressed
        commandOptionDown = sADMTracker->TestModifier(tracker,
            kADModKeyDownModifier);
        // checks if the (mac)COMMAND (win)CONTROL modifier is pressed
        commandControlDown = sADMTracker->TestModifier(tracker,
            kADMenuKeyDownModifier);

        pluginRef = sADMItem->GetPluginRef(item);
        menuListRef = sADMItem->GetList(item);

        if (menuListRef)
        {
            // if command is pressed, enable the first menu item,
            // otherwise disable
            sADMEEntry->Enable(sADMList->IndexEntry(menuListRef,
                0), commandOptionDown);
            // if command is pressed, enable the second menu item,
            // otherwise disable
            sADMEEntry->Enable(sADMList->IndexEntry(menuListRef,
                1), commandControlDown);
        }
    }

    doNotify = sADMItem->DefaultTrack(item, tracker);

    POP_GLOBALS

    return doNotify;
}

```

This covers the majority of the ADM Non-Modal Project ADM code.

You are encouraged to examine the other projects in the Illustrator SDK and to use the code there to jumpstart your own plug-in development process.

5

Using ADM with Adobe After Effects

Introduction

Adobe After Effects fully supports ADM. Both modal and non-modal dialogs are used. There are several examples of use of ADM in the After Effects SDK, available from <http://partners.adobe.com>.

Implementation notes

In the code below:

```
ADMDialogSuite *sADMDialog;  
error = SSPBasic->AcquireSuite(kADMDialogSuite, kADMDialogSuiteVersion2,  
&sADMDialog);  
if (error) goto . . . //handle error
```

the **sADMDialog** identifier is used to point to the functions in the **ADMDialogSuite**. After Effects plug-ins use other identifier names, but they are similar (for example, **adm_dialogP** instead of **sADMDialog**), so there won't be any confusion using the suite API description references in this manual.

Easy_Cheese Plug-in

The Easy_Cheese plug-in, found in the After Effects SDK, provides a good example of the use of ADM in After Effects.

6

The ADM Basic Suite

About the ADM Basic Suite

The ADM Basic suite provides four types of functions:

- [Resource Functions](#) - APIs that deal with platform resources.
- [User Interface Functions](#) - APIs that deal with the GUI.
- [Utility Functions](#) - APIs useful for miscellaneous tasks.
- [Contextual Menu Functions](#) - APIs for contextual menus.

Accessing the Suite

The ADM Basic suite is referred to as:

```
#define kADMBasicSuite "ADM Basic Suite"
```

with the version constants:

```
#define kADMBasicSuiteVersion2      2
```

NOTE: Determine the suite version number you are using by examining the `ADMBasic.h` header file.

The suite is acquired as follows:

```
ADMBasicSuite *sADMBasic;  
error = SSPBasic->AcquireSuite(kADMBasicSuite, kADMBasicSuiteVersion2,  
    &sADMBasic);  
if (error) goto . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

ADM Basic Suite Functions

Resource Functions

sADMBasic->GetIndexString()

Read a string from a resource

```
int ASAPI (*GetIndexString)(SPPluginRef inPluginRef, ASInt32
inStringID, ASInt32 inStringIndex, char* outString, ASInt32
inMaxLen);
```

Description

The **GetIndexString()** function returns a platform string resource, pointed to by **outString**. The returned string is a C style string.

On the Macintosh, this function reads strings from a '**STR#**' string list resource. The strings are converted from Pascal to C strings. Under Windows, the function reads a string from a string resource. The **inStringID** is the '**STR#**' resource ID. The **inStringIndex** is the 1-based index of the string in the list. The **inStringID** and **inStringIndex** values are added together to produce the resource ID to be read.

NOTE: Because Macintosh string list resources are 1-based, use a dummy argument for **inStringID**.

Parameters

inPluginRef	Plug-in reference.
inStringID	The resource ID.
inStringIndex	1-based index of the string in the list.
outString	The returned string.
inMaxLen	Maximum number of characters to be read.

Returns

The number of characters actually read.

Example

```
kMenuStrings          <First Mac index is 1>
(kMenuStrings + 1)    "My Menu Item"
(kMenuStrings + 2)    "My Other Menu Item"
```

```
// this same code would work on any platform
#define kMaxMenuLen 32
int strLength;
char menuString[kMaxMenuLen]

strLength = sADMBasic->GetIndexString(message->d.plugin, kMenuStrings,
    1, menuString, kMaxMenuLen);
```

sADMBasic->SetPlatformCursor()

Set the cursor type

```
void ASAPI (*SetPlatformCursor)(SPPluginRef inPluginRef,
    ASInt32 inCursorID);
```

Description

The **SetPlatformCursor()** function sets the cursor to a given platform-specific cursor resource. This is a cross-platform cursor setting mechanism, but the resource type is different for Macintosh (**'CURS'**) and Windows (**CURS**).

Parameters

inPluginRef	Plug-in reference.
inCursorID	Cursor resource ID.

Returns

None.

User Interface Functions

User interface functions control basic user communications such as beeps, alerts, and tool tips.

NOTE: Tool tips provide information about the ADM item currently pointed to by the mouse cursor. A predefined string describing the item appears to the right of the item after the mouse is positioned over it for a few seconds. When the mouse is moved, the tool tip disappears. The strings to use for a dialog are defined with the [sADMDialog->LoadToolTips\(\)](#). For Illustrator and other Adobe tools, the tool title is used for the tool tip.

sADMBasic->AboutBox()

Create an Adobe plug-in About box

```
void ASAPI (*AboutBox)(SPPluginRef inPlugin, const char*
    inText1, const char* inText2);
```

Description

The **AboutBox()** function is for Adobe in-house use. Third-party plug-in developers should use the [sADMBasic->PluginAboutBox\(\)](#) function.

This function is used to create a simple fixed-sized, three-line, title-less About box for a plug-in. The first two lines are user-defined; the third line contains appropriate copyright notice similar to: ©1997-98, Adobe Systems, Inc.

Parameters

inPlugin	Plug-in reference.
inText1	First user-defined line of text.
inText2	Second user-defined line of text.

Returns

None.

See also

[sADMBasic->PluginAboutBox\(\)](#)

sADMBasic->AreToolTipsEnabled()

Determine whether ADM tool tips are on or off

```
ASBoolean ASAPI (*AreToolTipsEnabled) ( );
```

Description

The **AreToolTipsEnabled()** function is used to determine whether ADM tool tips are currently on or off.

Parameters

None.

Returns

true if tool tips are on; **false** otherwise.

See also

[sADMBasic->EnableToolTips\(\)](#)
[sADMBasic->HideToolTip\(\)](#)
[sADMBasic->ShowToolTip\(\)](#)
[sADMBasic->AreToolTipsSticky\(\)](#)

sADMBasic->AreToolTipsSticky()

Determine whether ADM tool tips are sticky

```
ASBoolean ASAPI (*AreToolTipsSticky)(ADMDialogRef inDialog);
```

Description

The **AreToolTipsSticky()** function is used to determine whether ADM tool tips are currently in a “sticky” state—meaning that a tool tip is already visible, so no delay is needed before displaying the next tip (if the user moves the cursor, requiring display of another tip).

Parameters

inDialog	Reference to an instance of a dialog.
-----------------	---------------------------------------

Returns

true if tool tips are sticky; **false** otherwise.

See also

[sADMBasic->EnableToolTips\(\)](#)
[sADMBasic->HideToolTip\(\)](#)
[sADMBasic->ShowToolTip\(\)](#)
[sADMBasic->AreToolTipsEnabled\(\)](#)

sADMBasic->Beep()

Do a simple system beep

```
void ASAPI (*Beep)();
```

Description

The **Beep()** function causes a simple system beep. The platform's standard beep function is called.

Parameters

None.

Returns

None.

sADMBasic->ChooseColor()

Invoke the standard platform color picker

```
ASBoolean ASAPI (*ChooseColor)(ASPoint inWhere, const  
ASRGBColor* inColor, ASRGBColor* outColor);
```

Description

The **ChooseColor()** function invokes the standard platform color picker. Position (0, 0) centers the dialog on the screen.

Parameters

inWhere	The position argument; determines the location of the dialog. Type: ASPoint (see <code>ASTypes.h</code>)
inColor	Allows for setting of initial “picked” color values. If the end-user exits from the color picker dialog without choosing a color, then false is returned and outColor should not be used. Type: ASRGBColor (see <code>ASTypes.h</code>)
outColor	Returns the selected color. Type: ASRGBColor (see <code>ASTypes.h</code>)

Returns

true if the new color should be used; **false** otherwise. If the end-user exits from the color picker dialog without choosing a color, then **false** is returned and **outColor** should not be used.

See also

[sADMBasic->ADMColorToRGBColor\(\)](#)

sADMBasic->EnableToolTips()

Turn ADM tool tips on and off

```
void ASAPI (*EnableToolTips) (ASBoolean inEnable);
```

Description

The **EnableToolTips()** function turns ADM tool tips on and off. When on, ADM tool tips appear next to an ADM Dialog item (when a tool tip string is available for the item).

Parameters

inEnable	Boolean indicating whether tool tips should be turned on (true) or off (false).
-----------------	---

Returns

None.

See also

[sADMBasic->HideToolTip\(\)](#)
[sADMBasic->ShowToolTip\(\)](#)

[sADMBasic->AreToolTipsEnabled\(\)](#)
[sADMBasic->AreToolTipsSticky\(\)](#)

sADMBasic->ErrorAlert()

Present an error dialog to the user

```
void ASAPI (*ErrorAlert)(const char* inErrorString);
```

Description

The **ErrorAlert()** function informs the user that an error occurred. The text **inErrorString** is displayed to the user with an **OK** button. The platform's icon for an error is displayed to the left of the string. This function uses [sADMDialog->Modal\(\)](#) to put up the alert, which causes it to be more expensive than [sADMBasic->LightweightErrorAlert\(\)](#).

Parameters

inErrorString	Error string text.
----------------------	--------------------

Returns

None.

See also

[sADMBasic->MessageAlert\(\)](#)
[sADMBasic->LightweightErrorAlert\(\)](#)
[sADMBasic->QuestionAlert\(\)](#)
[sADMBasic->SetAlertButtonText\(\)](#)
[sADMBasic->YesNoAlert\(\)](#)

sADMBasic->GetToolTipDelays()

Get the tool tip delay time and pop-up duration parameters

```
void ASAPI (*GetToolTipDelays)(ASInt32* outPopupDelay,  
ASInt32* outPopdownDelay);
```

Description

The **GetToolTipDelays()** function is used to get the tool tips delay time and pop-up duration parameters. The default popup delay setting is .5 seconds; the default popdown delay setting is 5 seconds.

NOTE: Currently there is no function that can be used to set these parameters.

Parameters

outPopupDelay	Returns the amount of time before a tool tips is displayed.
----------------------	---

outPopdownDelay	Returns the amount of time a tool tip remains displayed.
------------------------	--

Returns

None.

sADMBasic->HideToolTip()

Hide tool tip

```
void WINAPI (*HideToolTip)();
```

Description

The **HideToolTip()** hides a tool tip displayed by the [sADMBasic->ShowToolTip\(\)](#) function. This function does not affect the standard tool tip behavior.

Parameters

None.

Returns

None.

See also

[sADMBasic->EnableToolTips\(\)](#)
[sADMBasic->ShowToolTip\(\)](#)
[sADMBasic->AreToolTipsEnabled\(\)](#)
[sADMBasic->AreToolTipsSticky\(\)](#)

sADMBasic->LightweightErrorAlert()

Present a lightweight error

```
void WINAPI (*LightweightErrorAlert)(const char*
inErrorString);
```

Description

The **LightweightErrorAlert()** function is a lightweight version of [sADMBasic->ErrorAlert\(\)](#). It indicates an error with a text message and a beep. This function uses the native platform mechanism for communicating an alert. On Windows it uses **MessageBox**, and on Mac it uses **CautionAlert**.

[sADMBasic->ErrorAlert\(\)](#) uses [sADMDialog->Modal\(\)](#) to put up the alert, which is a more expensive operation.

Parameters

inErrorString	Error string text.
----------------------	--------------------

Returns

None.

See also

[sADMBasic->ErrorAlert\(\)](#)
[sADMBasic->MessageAlert\(\)](#)
[sADMBasic->QuestionAlert\(\)](#)
[sADMBasic->SetAlertButtonText\(\)](#)
[sADMBasic->YesNoAlert\(\)](#)

sADMBasic->MessageAlert()

Present an information dialog to the user

```
void ASAPI (*MessageAlert)(const char* inMessageString);
```

Description

The **MessageAlert()** function displays an informational message to the user. The text **inMessageString** is displayed to the user with an **OK** button. The platform's icon for a message will be displayed to the left of the string. This function uses [sADMDialog->Modal\(\)](#) to put up the alert.

Parameters

inMessageString	Message string text.
------------------------	----------------------

Returns

None.

See also

[sADMBasic->ErrorAlert\(\)](#)
[sADMBasic->LightweightErrorAlert\(\)](#)
[sADMBasic->QuestionAlert\(\)](#)

sADMBasic->PluginAboutBox()

Present an information screen to the user

```
void ASAPI (*PluginAboutBox)(const char* inTitle, const char* inText);
```

Description

The **PluginAboutBox()** function displays an informational message to the user that describes the plug-in. The text **inTitle** is displayed in the title bar and the text **inText** is displayed in the dialog. The box grows vertically to hold the text.

Parameters

inTitle	Text displayed in the title bar of the dialog.
inText	Text displayed in the dialog.

Returns

None.

See also

[sADMBasic->AboutBox\(\)](#)

sADMBasic->QuestionAlert()

Present a question dialog to the user

```
ADMAnswer ASAPI (*QuestionAlert)(const char*
inQuestionString);
```

Description

The **QuestionAlert()** function asks the user a “yes” or “no” question. This function uses [sADMDialog->Modal\(\)](#) to put up the alert. The text **inQuestionString** is displayed to the user along with **Yes**, **No**, and **Cancel** buttons. The return value is one of type **ADMAnswer**:

```
typedef enum
{
    kADMNoAnswer = 0,
    kADMYesAnswer,
    kADMCancelAnswer,
    kADMDummyAnswer = 0xFFFFFFFF
}
ADMAnswer;
```

NOTE: ADM follows the C language convention in enumerated lists—i.e., when an enumerated list starts with an assignment value, subsequent entries are given consecutively numbered values. Thus, **kADMYesAnswer** = 1, **kADMCancelAnswer** = 2, etc., until the list is exhausted or a new value is assigned to an entry.

Parameters

inQuestionString	Alert string text.
-------------------------	--------------------

Returns

Enumerated value of type **ADMAnswer**.

See also

[sADMBasic->ErrorAlert\(\)](#)
[sADMBasic->LightweightErrorAlert\(\)](#)
[sADMBasic->MessageAlert\(\)](#)
[sADMBasic->SetAlertButtonText\(\)](#)
[sADMBasic->YesNoAlert\(\)](#)

sADMBasic->SetAlertButtonText()

Set the button text for an Alert dialog

```
void ASAPI (*SetAlertButtonText)(const char* inLeftChoice,  
const char* inMiddleChoice, const char* inRightChoice);
```

Description

The **SetAlertButtonText()** function sets the text labels for the buttons of an [sADMBasic->QuestionAlert\(\)](#) or [sADMBasic->YesNoAlert\(\)](#) dialog. This should be called before [sADMBasic->QuestionAlert\(\)](#) or [sADMBasic->YesNoAlert\(\)](#) is called.

NOTE: Default values are reset automatically after the alert is called. The sequence is: set alert, call the alert, automatic reset, set the alert, call the alert, automatic reset, and so on.

Parameters

inLeftChoice	Text for left button. Default: Yes.
inMiddleChoice	Text for middle button. Default: No.
inRightChoice	Text for right button. Default: Cancel.

Returns

See [sADMBasic->QuestionAlert\(\)](#) or [sADMBasic->YesNoAlert\(\)](#).

See also

[sADMBasic->QuestionAlert\(\)](#)
[sADMBasic->YesNoAlert\(\)](#)

sADMBasic->ShowToolTip()

Show ADM tool tips

```
void ASAPI (*ShowToolTip)(const ASPoint* inWhere, const char*  
inTip);
```

Description

The **ShowToolTip()** function shows an ADM tool tip at the position **inWhere** using the text value **inTip**. The tip disappears after 5 seconds unless [sADMBasic->HideToolTip\(\)](#) is called.

Parameters

inWhere	Position where the tool tip is to be displayed. Type: ASPoint (see <code>ASTypes.h</code>)
inTip	Tool tip text.

See also

[sADMBasic->EnableToolTips\(\)](#)
[sADMBasic->HideToolTip\(\)](#)
[sADMBasic->AreToolTipsEnabled\(\)](#)
[sADMBasic->AreToolTipsSticky\(\)](#)

sADMBasic->StandardGetFileDialog()

Open standard file open dialog

```
ASBoolean ASAPI (*StandardGetFileDialog)(const char*
inMessage, const ADMPatformFileTypesSpecification3* inFilter,
const SPPlatformFileSpecification* inStartingDir, const char*
inStartingFile, SPPlatformFileSpecification* outResult);
```

Description

The **StandardGetFileDialog()** function opens the platform standard open file dialog. The text **inMessage** displays at the top of the dialog. **inFilter** determines what information is displayed in the dialog. **inStartingDir** selects the starting directory to be displayed and **inStartingFile** selects the starting file. The selected file is returned in **outResult**.

Windows filters are a string specification which designates files for display in the file dialog based upon their extensions. A filter may have several choices for display groups (such as Executable Files, Document Files, All Files, etc.), each of which specifies one or more extensions which should be included. For example:

Executable Files (*.exe, *.dll, *.aip, *.apl)

Document Files (*.psd, *.jpg, *.gif)

All Files

The user can choose one of these from the dropdown menu in the dialog box to select what types of files the dialog will display. Each line is a “group” of extensions, and the set of lines is the entire “filter.”

To specify a Windows filter, you create a string containing groups, each of which contains a “title” string and a “specification” string separated by \0 and terminated with \0. Multiple extensions in the specification string are separated by semicolons. Groups are concatenated to create a filter, which is terminated by an additional \0, resulting in there being two \0 characters at the end of the filter.

The filter string for the above example consists of:

```
static ADMPlatformFileTypesSpecification3 sampleFilter = {
    "Executable Files (*.exe, *.dll, *.aip, *.apl)\0"
    "*.exe;*.dll;*.aip;*.apl\0"
    "Document Files (*.psd, *.jpg, *.gif)\0"
    "*.psd;*.jpg;*.gif\0"
    "All Files\0"
    ".*\0"
    "\0"
};
```

On the Mac, **StandardGetFileDialog()** uses the Mac definition of the **ADMPlatformFileTypesSpecification** structure as filter, which is:

```
typedef struct
{
    unsigned long *types;
    short numTypes;
} ADMPlatformFileTypesSpecification;

typedef struct
{
    unsigned long *types;
    short numTypes;
    char filter[256];
} ADMPlatformFileTypesSpecification3;
```

Where:

- **types** is a pointer to an array of file types,
- **numTypes** contains the number of file types in **types**
- **filter** contains a list of extensions. The function first checks for the file types.

Conditions:

- If the file types have not been provided, it checks for the extensions in **filter**.
- If the file types are found, the filter is ignored.
- The filter can contain series of extensions, separated by commas or blanks. A wild card can be used only before the ".". For example, don't use ***.p***—use ***.pdf**.

For example:

```
ADMPlatformFileTypesSpecification3 theFilter;
long types[2];
types[0] = 'APPL';
types[1] = 'TEXT';
theFilter.types = types;
theFilter.numTypes = 2;
```

sADMBasic->StandardPutFileDialog() has **inFilter** as a parameter but doesn't use it. It was added for Windows compatibility.

Parameters

inMessage	Text displayed at the top of the dialog.
inFilter	Filter that determines what information is displayed. On the Macintosh platform you can list files, types, and/or extensions. On the Windows platform you can list files with extensions. Type: ADMPatformFileTypesSpecification3 (see ADMBasic.h)
inStartingDir	Starting directory. Type: SPPlatformFileSpecification (see ADMBasic.h)
inStartingFile	Starting file.
outResult	File selected by user. Type: SPPlatformFileSpecification (see ADMBasic.h)

Returns

true if file was found and returned; **false** otherwise.

See also

[sADMBasic->StandardPutFileDialog\(\)](#)
[sADMBasic->StandardGetDirectoryDialog\(\)](#)

sADMBasic->StandardGetDirectoryDialog()

Open standard
directory select
dialog

```
ASBoolean ASAPI (*StandardGetDirectoryDialog)(const char*
inMessage, const SPPlatformFileSpecification* inStartingDir,
SPPlatformFileSpecification* outResult);
```

Description

The **StandardGetDirectoryDialog()** function opens the platform standard open directory dialog. **inStartingDir** selects the starting directory. The selected directory is returned in **outResult**.

Parameters

inMessage	Text displayed at the top of the dialog.
inStartingDir	Starting directory. Type: SPPlatformFileSpecification (see ADMBasic.h)

outResult	Directory selected by user. Type: SPPlatformFileSpecification (see <code>ADMBasic.h</code>)
------------------	--

Returns

true if directory was found and returned; **false** otherwise.

See also

[sADMBasic->StandardGetFileDialog\(\)](#)
[sADMBasic->StandardPutFileDialog\(\)](#)

sADMBasic->StandardPutFileDialog()

Open standard file save dialog

```
ASBoolean ASAPI (*StandardPutFileDialog)(const char*
inMessage, const ADMPatformFileTypesSpecification3* inFilter,
const SPPlatformFileSpecification* inStartingDir, const char*
inStartingFile, SPPlatformFileSpecification* outResult);
```

Description

The **StandardPutFileDialog()** opens the platform standard save file dialog. The text **inMessage** displays at the top of the dialog. **inFilter** determines what information is displayed in the dialog. **inStartingDir** selects the starting directory to be displayed and **inStartingFile** selects the starting file. The selected file is returned in **outResult**.

Parameters

inMessage	Text displayed at the top of the dialog.
inFilter	Filter that determines what information is displayed. On the Macintosh platform you can list files, types, and/or extensions. On the Windows platform you can list files with extensions. Type: ADMPatformFileTypesSpecification3 (see <code>ADMBasic.h</code>)
inStartingDir	Starting directory. Type: SPPlatformFileSpecification (see <code>ADMBasic.h</code>)
inStartingFile	Starting file.
outResult	File selected by user. Type: SPPlatformFileSpecification (see <code>ADMBasic.h</code>)

Returns

true if file was saved; **false** otherwise.

See also

[sADMBasic->StandardGetFileDialog\(\)](#)
[sADMBasic->StandardGetDirectoryDialog\(\)](#)

sADMBasic->YesNoAlert()

Create yes/no Alert dialog

```
ADMAnswer ASAPI (*YesNoAlert)(const char* inQuestionString);
```

Description

The **YesNoAlert()** function creates a simple “yes” or “no” type alert dialog. This function uses [sADMDialog->Modal\(\)](#) to put up the alert. The text **inQuestionString** is displayed to the user along with **Yes**, **No**, and **Cancel** buttons. The return value is one of type **ADMAnswer**:

```
typedef enum
{
    kADMNoAnswer = 0,
    kADMYesAnswer,
    kADMCancelAnswer,
    kADMDummyAnswer = 0xFFFFFFFF
}
ADMAnswer;
```

NOTE: ADM follows the C language convention in enumerated lists—i.e., when an enumerated list starts with an assignment value, subsequent entries are given consecutively numbered values. Thus, **kADMYesAnswer** = 1, **kADMCancelAnswer** = 2, etc., until the list is exhausted or a new value is assigned to an entry.

Parameters

inQuestionString	Alert string text.
-------------------------	--------------------

Returns

Enumerated value of type **ADMAnswer**.

See also

[sADMBasic->ErrorAlert\(\)](#)
[sADMBasic->LightweightErrorAlert\(\)](#)
[sADMBasic->QuestionAlert\(\)](#)
[sADMBasic->MessageAlert\(\)](#)
[sADMBasic->SetAlertButtonText\(\)](#)

Utility Functions

sADMBasic->ADMColorToRGBColor()

Convert an ADM color to an RGB value

```
ASBoolean ASAPI (*ADMColorToRGBColor)(ADMColor inADMColor,
ASRGBColor* outRGBColor);
```

Description

The **ADMColorToRGBColor()** function converts an ADM standard color to an RGB value.

Parameters

inADMColor	ADM color for conversion. Type: ADMColor (see ADMTypes.h)
outRGBColor	ADM color converted to RGB value. Type: ASRGBColor (see ASTypes.h)

Returns

true if the conversion was successful; **false** otherwise.

See also

[sADMBasic->ChooseColor\(\)](#)

sADMBasic->GetAppFPS()

Get the application's FPS

```
float ASAPI (*GetAppFPS)();
```

Description

The **GetAppFPS()** returns the application's frames-per-second (FPS) value.

Parameters

None.

Returns

FPS value.

See also

[sADMBasic->SetAppFPS\(\)](#)

sADMBasic->GetAppUnits()

Get the units used by the host application

```
ADMUnits WINAPI (*GetAppUnits)(ADMUnits inAppUnits);
```

Description

The **GetAppUnits()** function returns the units currently being used by the host application (points, picas, inches, etc.) for **inAppUnits**. **inAppUnits** can be one of 10 enumerated values of type **ADMUnits**.

ADM provides a system for storing 10 distinct unit preferences that may be used for various purposes within an application or plug-in. The first argument to [sADMBasic->SetAppUnits\(\)](#), **inAppUnits**, is used to specify which unit preference is being set (#1-10), while the second argument, **inActualUnits**, is used to set the units for that particular unit preference. The **GetAppUnits()** function takes only an **inAppUnits** argument.

Parameters

inAppUnits	Unit preference (#1-10) for which inActualUnits (see sADMBasic->SetAppUnits()) is being requested. One of 10 enumerated values of type ADMUnits (see <code>ADMTypes.h</code>).
-------------------	---

Returns

The **inActualUnits** (see [sADMBasic->SetAppUnits\(\)](#)) units currently being used by the application for **inAppUnits**. Must be one of the enumerated values of type **ADMUnits**, as defined in `ADMTypes.h`.

See also

[sADMBasic->SetAppUnits\(\)](#)

sADMBasic->GetDefaultIncrements()

Get default increments for cursor movement in spinners, sliders, and scroll bars

```
void WINAPI (*GetDefaultIncrements)(ADMUnits inUnits, float* outSmallIncrement, float* outLargeIncrement);
```

Description

The **GetDefaultIncrements()** function obtains the default cursor movement increments set by [sADMBasic->SetDefaultIncrements\(\)](#).

Parameters

inUnits	Units for which the incremental values are being retrieved (e.g., kADMPixelUnits). Type: ADMUnits (see ADMTypes.h)
outSmallIncrement	Small increment for inUnits .
outLargeIncrement	Large increment for inUnits .

Returns

None.

See also

[sADMBasic->SetDefaultIncrements\(\)](#)

sADMBasic->GetLastADMErrror()

Get the last ADM error data

```
ASBoolean ASAPI (*GetLastADMErrror)(ASInt32* outError, ASInt32*
outErrorData);
```

Description

The **GetLastADMErrror()** function is used to retrieve relevant information about why a dialog failed to load after using [sADMDialog->Create\(\)](#), [sADMDialog->Modal\(\)](#), or [sADMDialog->CreateGroupInDialog\(\)](#).

Parameters

outError	Type of error. The error codes returned can be found in different locations in the header files. Some are defined in ADMDialog.h (see Dialog Error Codes). Others are defined in ASTypes.h . Also see ADMCustomerResource.h and ADMDialogGroup.h .
outErrorData	Data associated with outError (e.g., dialog reference).

Returns

The return value is always **false**. It has no significance.

sADMBasic->GetNumbersArePoints()

Determine whether numbers are points

```
ASBoolean ASAPI (*GetNumbersArePoints)();
```

Description

The **GetNumbersArePoints()** function determines whether numbers entered without a units specifier are in points (72 pts./in.). This function is useful for applications that do not default to the use of points. For example, Illustrator (2D drawing-oriented) does use point measurements, while Premiere (time-based oriented) does not.

Parameters

None.

Returns

true if numbers entered without a units specifier are in points; **false** otherwise.

See also

[sADMBasic->SetNumbersArePoints\(\)](#)

sADMBasic->GetPaletteLayoutBounds()

Get the dimensions of the host application palette layout bounds

```
ASBoolean ASAPI (*GetPaletteLayoutBounds)(ASRect*
outDimensions);
```

Description

The **GetPaletteLayoutBounds()** function returns the dimensions of the host application screen boundaries (the inset of a standard application window). This is the area is available for use in placing a plug-in dialog screen.

Parameters

outDimensions	Dimensions of the host application screen boundaries. Type: ASRect (see ASTypes.h)
----------------------	--

Returns

true if the dimensions are successfully obtained; **false** otherwise.

See also

[sADMBasic->GetWorkspaceBounds\(\)](#)

sADMBasic->GetScreenDimensions()

Get the dimensions of the screen

```
ASBoolean ASAPI (*GetScreenDimensions) (const ASPoint*
inPoint, ASRect* outDimensions);
```

Description

The **GetScreenDimensions()** function returns the dimensions of the primary screen. Finds the bounds of the screen containing **inPoint**. Designed to work with both Windows and Mac platforms and supports multiple monitor setups.

Parameters

inPoint	A point on the screen. Type: ASPoint (see ASTypes.h)
outDimensions	Screen dimensions. Type: ASRect (see ASTypes.h)

Returns

true if **inPoint** is on any screen; **false** otherwise.

sADMBasic->GetWorkspaceBounds()

Get the dimensions of the available screen workspace

```
ASBoolean ASAPI (*GetWorkspaceBounds)(ASRect* outDimensions);
```

Description

The **GetWorkspaceBounds()** function returns the dimensions of the primary screen minus the system menus, task bars, etc.

Parameters

outDimensions	Workspace dimensions. Type: ASRect (see ASTypes.h)
----------------------	---

Returns

true if dimensions successfully returned; **false** otherwise.

See also

[sADMBasic->GetPaletteLayoutBounds\(\)](#)

sADMBasic->LookUpZString()

Look up the translation of a ZString literal

```
ASBoolean ASAPI (*LookUpZString)(SPPluginRef inPluginRef,
const char* inString, char* outString, ASUInt32*
ioBufferSize);
```

Description

The **LookUpZString()** function looks up the translation of a ZString literal.

Parameters

inPluginRef	Plug-in reference.
inString	ZString.
outString	Translated ZString.
ioBufferSize	Input/output pointer. On input, it is the size of the buffer; on output, it is the size required to hold the entire string—that is, the minimum buffer size which will not truncate the string.

Returns

true if string successfully returned; **false** otherwise.

sADMBasic->SetAppFPS()

Set the application's FPS

```
void ASAPI (*SetAppFPS)(float inFPS);
```

Description

The **SetAppFPS()** function sets the application's frames-per-second (FPS).

Parameters

inFPS	FPS value.
--------------	------------

Returns

None.

See also

[sADMBasic->GetAppFPS\(\)](#)

sADMBasic->SetAppUnits()

Set the units being used by the application

```
void ASAPI (*SetAppUnits)(ADMUnits inAppUnits, ADMUnits
inActualUnits);
```

Description

The **SetAppUnits()** function sets the units being used by the application.

ADM provides a system for storing 10 distinct unit preferences that may be used for various purposes within an application or plug-in. The first argument to **SetAppUnits()**, **inAppUnits**, is used to specify which unit preference is being set (#1-10), while the second argument, **inActualUnits**, is used to set the units for that

particular unit preference. The [sADMBasic->GetAppUnits\(\)](#) function takes only an **inAppUnits** argument and returns the **inActualUnits**.

As an example of how to use these functions, an application might use app unit #1 as the default units for most measurements and app unit #2 as the default units for typographic measurements. At any one time, app units #1 may be inches and app units #2 may be points, or app units #1 may be centimeters, and app units #2 may be millimeters. In the application, a field with a typographic measurement may set itself to app units #2, and display in the real units that are currently assigned to app units #2.

Parameters

inAppUnits	Unit preference. One of 10 enumerated values of type ADMUnits (see <code>ADMTypes.h</code>).
inActualUnits	Units for the inAppUnits unit preference. Type: ADMUnits (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMBasic->GetAppUnits\(\)](#)

sADMBasic->SetDefaultIncrements()

Sets the defaults for spinner, slider, and scroll bar cursor movements

```
void ASAPI (*SetDefaultIncrements)(kADMPixelUnits inUnits,
float inSmallIncrement, float inLargeIncrement);
```

Description

The **SetDefaultIncrements()** function sets the small and large increments for spinner, slider, and scroll bar cursor movement.

Parameters

inUnits	Units for which the incremental values are being set (e.g., kADMPixelUnits). Type: ADMUnits (see <code>ADMTypes.h</code>)
inSmallIncrement	Small increment for inUnits .
inLargeIncrement	Large increment for inUnits .

Returns

None.

See also

[sADMBasic->GetDefaultIncrements\(\)](#)

sADMBasic->SetNumbersArePoints()

Sets the unit for numbers as points

```
void ASAPI (*SetNumbersArePoints)(ASBoolean inPoints);
```

Description

The **SetNumbersArePoints()** function is used to set whether numbers entered without a units specifier are in points (72 pts./in.). Useful for applications that do not default to the use of points. For example, Illustrator (2D drawing oriented) does use point measurements, but Premiere (time-based oriented) does not.

Parameters

inPoints	If true , points will be the default unit of measurement in the application. If false , they will not be.
-----------------	---

Returns

None.

See also

[sADMBasic->GetNumbersArePoints\(\)](#)

sADMBasic->StringToValue()

Convert a string to a value

```
ASBoolean ASAPI (*StringToValue)(const char* inText, float* outValue, ADMUnits inUnits);
```

Description

The **StringToValue()** function converts between C string **inText** and float **outValue** in points. If indicated, **inUnits** is used to scale the result.

Parameters

inText	Floating point value to convert to a C string.
outValue	inValue converted to a C string.
inUnits	Used to scale inValue after conversion. Type: ADMTUnits (see ADMTTypes.h)

Returns

true if the conversion was successful; **false** otherwise.

See also[sADMBasic->ValueToString\(\)](#)

sADMBasic->ValueToString()

Convert a value to a string

```
void ASAPI (*ValueToString)(float inValue, char* outText,  
ASInt32 inMaxLen, ADMUnits inUnits, ASInt32 inPrecision,  
ASBoolean inAlwaysAppendUnits);
```

Description

The **ValueToString()** function converts between float **inValue** and C string **outText**. If indicated, **inUnits** is used to scale the value before it is converted. **inPrecision** indicates the maximum number of decimal places that are used in the final string. If the units value should be appended to the string, set **inAlwaysAppendUnits** to **true**.

Units values and their scale are given in **StringToValue()** above.

Parameters

inValue	Floating point value to convert to a C string.
outText	inValue converted to a C string.
inMaxLen	Maximum length for string outText .
inUnits	Used to scale inValue before conversion. Type: ADMTUnits (see ADMTTypes.h)
inPrecision	Maximum number of decimal places that are used in the final string.
inAlwaysAppendUnits	If true , units are appended to string.

Returns

None.

See also[sADMBasic->StringToValue\(\)](#)

Contextual Menu Functions

These functions are used to create pop-up contextual menus. Contextual menus can be used to display a menu when the cursor is over a particular location and a combination of mouse states or modifier keys can be used to select an option. For example, pressing and holding the mouse down in an Internet browser window opens

a pop-up menu with **Back** and **Forward** options. Most likely, these functions will be used by the host application rather than a plug-in. Contextual menus must be destroyed when done.

sADMBasic->CreateMenu()

Create a contextual ADM menu

```
ASErr ASAPI (*CreateMenu)(ADMListRef* outMenu);
```

The **CreateMenu()** function creates a contextual menu. It allows the creation of a list without an associated item. The list reference returned in **outMenu** can be initialized with the standard functions in the ADM List and ADM Entry suites.

Parameters

outMenu	The new contextual menu.
----------------	--------------------------

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

See also

[sADMBasic->DestroyMenu\(\)](#)
[sADMBasic->DisplayMenu\(\)](#)

sADMBasic->DestroyMenu()

Destroy a contextual ADM menu

```
ASErr ASAPI (*DestroyMenu)(ADMListRef inMenu);
```

Description

The **DestroyMenu()** function destroys the referenced contextual menu.

Parameters

inMenu	The contextual menu to destroy.
---------------	---------------------------------

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes. See [Appendix D](#) for a list of ADM error codes.

See also

[sADMBasic->CreateMenu\(\)](#)
[sADMBasic->DisplayMenu\(\)](#)

sADMBasic->DisplayMenu()

Display a contextual ADM menu

```
ASErr ASAPI (*DisplayMenu)(ADMListRef inMenu, ADMDialogRef  
inDialog, ASPoint inDialogPoint);
```

Description

The **DisplayMenu()** function displays **inMenu** over **inDialog** at **inDialogPoint**.

Parameters

inMenu	Contextual menu to display.
inDialog	Dialog over which to display inMenu .
inDialogPoint	Point within inDialog at which to display inMenu . Type: ASPoint (see ASTypes.h)

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

See also

[sADMBasic->CreateMenu\(\)](#)
[sADMBasic->DestroyMenu\(\)](#)

7

The ADM Dialog Suite

About the ADM Dialog Suite

The ADM Dialog suite allows you to create and access ADM Dialog objects. Many of the functions are those common to all ADM objects, such as text access functions. Others are unique to dialogs—for instance, setting minimum and maximum sizes for resizable dialogs. This function reference builds on ideas established in the [Chapter 1, “ADM Overview”](#).

Accessing the Suite

The ADM Dialog suite is referred to as:

```
#define kADMDialogSuite "ADM Dialog Suite"
```

with the version constant:

```
#define kADMDialogSuiteVersion2 2
```

NOTE: Determine the suite version number you are using by examining the `ADMDialog.h` header file.

The suite is acquired as follows:

```
ADMDialogSuite *sADMDialog;  
error = sSPBasic->AcquireSuite(kADMDialogSuite, kADMDialogSuiteVersion2,  
    &sADMDialog);  
if (error) goto . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

Dialog Basics: Styles

ADM supports several types of dialog styles. Each dialog type is assigned a value as follows:

```
typedef enum  
{  
    kADModalDialogStyle = 0,  
    kADAlertDialogStyle = 1,  
    kADMFloatingDialogStyle = 2,  
    kADMTabbedFloatingDialogStyle = 3,  
    kADMResizingFloatingDialogStyle = 4,  
    kADMTabbedResizingFloatingDialogStyle = 5,  
    kADMPopupDialogStyle = 6,  
    kADMNoCloseFloatingDialogStyle = 7,  
}
```

```

    kADMSystemAlertDialogStyle = 8,
    kADMPopupControlDialogStyle = 9,
    kADMResizingModalDialogStyle = 10,
    kADMLeftSidedFloatingDialogStyle = 11,
    kADMLeftSidedNoCloseFloatingDialogStyle = 12,
    kADMNoTitleDockFloatingDialogStyle = 13,
    kADMHostDefinedDialogStyle = 65536,
    kADMDummyDialogStyle = 0xFFFFFFFF
}
ADMDialogStyle;

```

NOTE: Style constants above FFFF are reserved for host application use.

The valid modal dialog styles are:

```

typedef enum
{
    kADMModalDialogStyle = 0,
    kADMAlerDialogStyle = 1,
    kADMSystemAlertDialogStyle = 8,
    kADMDummyDialogStyle = 0xFFFFFFFF
}
ADMDialogStyle;

```

The modal, alert, and system alert styles (0, 1, 8) are for modal dialogs only; all others are for non-modal ones. Tabbed floating dialogs can be docked together. A resizable dialog can be resized according to the platform user interface guidelines. More complete descriptions of the dialog styles are given in [Chapter 1, “ADM Overview”](#).

Dialog Basics: Standard Dialog Item IDs

ADM provides many types of dialog items as outlined in [Chapter 1, “ADM Overview”](#). Each standard dialog item is assigned an ID as follows:

```

typedef enum
{
    kADMUniqueItemID = 0,
    kADMFirstItemID = -1,
    kADMLastItemID = -2,
    kADMDefaultItemID = -3,
    kADMCancelItemID = -4,
    kADMMenuItemID = -5,
    kADMResizeItemID = -6,
    kADMPrivateUniqueItemID = -7,
    kADMFirstUnusedPrivateItemID = -8,
    kADMDummyItemID = 0xFFFFFFFF
}
ADMStandardDialogItemID;

```


Dialog Basics: Callbacks

If default operation is not desired, most ADM dialog items support programmer-supplied callbacks, listed below and defined in `ADMDialog.h`. As shown below, these procedures include the dialog Init proc, the Draw proc, the Tracker proc, the Notify proc, and the dialog Destroy proc. The timer and the timer abort procedures are used to attach timers to dialogs. In each case, an **ADMDialogRef** variable (**inDialog**) specifies the dialog object on which the proc is to act.

```
typedef ASErr ASAPI (*ADMDialogInitProc)(ADMDialogRef inDialog);
typedef void ASAPI (*ADMDialogDrawProc)(ADMDialogRef inDialog,
    ADMDrawerRef inDrawer);
typedef ASBoolean ASAPI (*ADMDialogTrackProc)(ADMDialogRef inDialog,
    ADMTrackerRef inTracker);
typedef void ASAPI (*ADMDialogNotifyProc)(ADMDialogRef inDialog,
    ADMNotifierRef inNotifier);
typedef void ASAPI (*ADMDialogDestroyProc)(ADMDialogRef inDialog);
typedef ASBoolean ASAPI (*ADMDialogTimerProc)(ADMDialogRef inDialog,
    ADMTimerRef inTimerID);
typedef void ASAPI (*ADMDialogTimerAbortProc)(ADMDialogRef inDialog,
    ADMTimerRef inTimerID, ADMAction inAbortAction);
```

In the code above, **ADMDialogRef inDialog** is a reference to the object for which the procedure is being called and the additional reference argument (**inDrawer**, **inTracker**, **inNotifier**, **inTimerID**) is a reference to the action to be performed.

ADM Dialog Suite Functions

sADMDialog->AbortTimer()

Abort a timer

```
void ASAPI (*AbortTimer)(ADMDialogRef inDialog, ADMTimerRef
inTimerID);
```

Description

The **AbortTimer()** function aborts a timer procedure. It is used to destroy a dialog before the timer expires.

Parameters

inDialog	An ADM dialog.
inTimerID	Timer ID associated with inDialog . Type: ADMTimerRef (see <code>ADMTypes.h</code>)

Returns

None.

See also[sADMDialog->CreateTimer\(\)](#)**sADMDialog->Activate()**

Make a dialog active or inactive

```
void ASAPI (*Activate)(ADMDialogRef inDialog, ASBoolean
inActivate);
```

Description

The **Activate()** function activates or deactivates a floating dialog. An active dialog is the front-most dialog. When a modal dialog is visible, only it is visible.

Parameters

inDialog	An ADM dialog.
inActivate	true activates inDialog ; false deactivates it.

Returns

None.

See also[sADMDialog->IsActive\(\)](#)**sADMDialog->AdjustItemTabOrder()**

Adjust the item tab order in a dialog

```
ASErr ASAPI (*AdjustItemTabOrder)(ADMDialogRef inDialog,
ADMItemRef inItem, ASInt32 position);
```

Description

The **AdjustItemTabOrder()** function repositions the tab order of **inItem**. **inItem** should be already created.

Parameters

inDialog	An ADM dialog.
inItem	The item whose tab order is to be repositioned.
position	The new tab order position for inItem .

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

sADMDialog->Create()

Create a new ADM modeless dialog

```
ADMDialogRef ASAPI (*Create)(SPPluginRef inPluginRef, const
char* inName, ASInt32 inDialogID, ADMDialogStyle
inDialogStyle, ADMDialogInitProc inInitProc, ADMUserData
inUserData, ASInt32 inOptions);
```

Description

The **Create()** function creates a new ADM modeless or tabbed floating dialog. The **inPluginRef** is the plug-in creating the dialog and identifies the location of the dialog resource. Use the reference passed in the **SPMessageData** data structure when the plug-in is loaded. **inName** is the internal identifier for the dialog. It is the only ID that you can guarantee is unique, assuming that you use a unique name. **inDialogID** is the platform native resource ID to use in creating the dialog. ADM reads the resource correctly and creates the window and any dialog items. **inDialogStyle** indicates the type of dialog. **inInitProc** is a callback function you provide that is called after ADM has created the dialog. The **inUserData** argument is a 4-byte value that ADM keeps with the dialog; you can pass a pointer to a block of memory or some other value and retrieve it later using the [sADMDialog->GetUserData\(\)](#) function. The returned value is a reference to the created dialog and is used as an argument in many of the other ADM Dialog suite APIs. A dialog created with the **Create()** function is destroyed with the [sADMDialog->Destroy\(\)](#) function.

The **ADMDialogInitProc** is defined as:

```
typedef ASErr ASAPI (*ADMDialogInitProc)(ADMDialogRef dialog);
ASErr ASAPI ModalDlgInitProc(ADMDialogRef dialog)
{
    ASErr error = kNoErr;
    ADMItemRef dlgItem;

    // We want to set an OK button handler.
    dlgItem = sADMDialog->GetItem(dialog, kOKPushButton);
    sADMItem->SetNotifyProc(dlgItem,
        DlgOKButtonCallBackProc);

    // To set the state of a control, use the SetXXXValue() functions,
    // for instance, use the SetBooleanValue() for a check box.
    dlgItem = sADMDialog->GetItem(dialog, kShowAlertFrameCheckBox);
    sADMItem->SetBooleanValue(dlgItem,true);

    return kNoErr;
}
```

Parameters

inPluginRef	Plug-in reference.
--------------------	--------------------

inName	Internal identifier for the dialog. Must be unique. The suggested convention is to use a leading prefix of your company's name, as in MyCompanydialogxxxx .
inDialogID	Platform native resource ID. This ID refers to the platform-specific code provided to ADM to create the dialog on either the Macintosh or PC.
inDialogStyle	Style of dialog. See Dialog Basics: Styles . Type: ADMDialogStyle (see <code>ADMDialog.h</code>)
inInitProc	A callback with the following signature (see <code>ADMDialog.h</code>): ADMDialogInitProc (ADMDialogRef inDialog); Use to further initialize or customize the dialog. If special initialization is not required, pass NULL . Use inDialog to access the dialog items that need initialization. If inInitProc returns an error, the dialog will not be created. See Appendix D for a list of error codes that can be returned.
inUserData	A 4-byte value that ADM keeps with the dialog; you can pass a pointer to a block of memory or some other value and retrieve it later using the sADMDialog->GetUserData() function. Type: ADMUserData (see <code>ADMTypes.h</code>)
inOptions	Special options. If no special options are required, pass 0 (zero). See Possible values for inOptions .

TABLE 7.1 Possible values for *inOptions*

Flag	Meaning
kADMTabbedDialogShowsCycleOption (1L << 0)	This is for palettes that support a zoom cycle state. The cycle button allows you to cycle for different palette sizes (states) by clicking once on the cycle button instead of double clicking on the drag bar or tab. The cycle button is the little double arrow (one points up, the other down) that is drawn before the tab title. An example would be the Gradient palette in Illustrator.
kADMPassMouseDownEventInModalDialogOption (1L << 1)	Used to allow modal dialogs to pass mouse down events through to the user dialog tracker.
kReservedForUseByCustomResourcesOption (1L << 2)	Reserved.

TABLE 7.1 Possible values for *inOptions*

Flag	Meaning
<code>kADMKeypadEnterNotDefault (1L << 3)</code>	enter key does not activate default item.
<code>kADMCreateDialogItemsHiddenByDefault (1L << 4)</code>	Reduces flicker by creating items hidden.
<code>kADMForceRomanFontForDialog (1L << 5)</code>	Forces for all items within dialog, except as overridden.
<code>kADMTrackEnterBeforeDialogDoesOK (1L << 6)</code>	Track return and enter keys before the dialog treats the event as equivalent to pressing the OK button—and prevent that behavior if the tracker returns true . NOTE: By default, these keys cause text item trackers to commit their text and return true , so this option normally prevents the OK button behavior when the key is pressed within a text item. (This option is currently only relevant on Mac platform.)
<code>kADMModalDialogHasPaletteSystemControlsOption (1L << 7)</code>	0 by default. If set, ADM modal dialogs on Windows will have a close box on the top right-hand corner. There also is a host option that a user can use if all dialogs in the application need that behavior.
<code>kADMCreatePopupDialogAsFloatingClassOption (1L << 8)</code>	Helps make popup dialogs in palettes layer properly.
<code>1L << 28</code> and higher shifts	Reserved.

Returns

An ADM Dialog.

See also

[sADMDialog->Destroy\(\)](#)
[sADMDialog->Modal\(\)](#)
[sADMDialog->GetUserData\(\)](#)

Example

```

sADMDialog->Create(message->d.self, "ADMNonModalDialog",
kADMNonModalDialogID, kADMTabbedFloatingDialogStyle, ModalDlgInitProc,
nil, 0);

ASErr ASAPI ModalDlgInitProc(ADMDialogRef dialog)
{
    ASErr error = kNoErr;
    ADMItemRef dlgItem;

    // We want to set an OK button handler.
    dlgItem = sADMDialog->GetItem(dialog, kOKPushButton);
    sADMItem->SetNotifyProc(dlgItem, DlgOKButtonCallBackProc);

    // To set the state of a control, use the SetXXXValue() functions,
    // for instance, use the SetBooleanValue() for a check box.
    dlgItem = sADMDialog->GetItem(dialog, kShowAlertFrameCheckBox);
    sADMItem->SetBooleanValue(dlgItem,true);

    return kNoErr;
}

```

sADMDialog->CreateGroupInDialog()

Create a group in a dialog

```

ADMItemRef ASAPI (*CreateGroupInDialog)(ADMDialogRef inDialog,
SPPluginRef inPluginRef, const char* inName, ASInt32
inDialogID, ADMItemInitProc inInitProc, ADMUserData
inUserData, ASInt32 inOptions);

```

Description

The **CreateGroupInDialog()** function creates an ADM Item group (**kADMItemGroupType**) in a dialog.

Parameters

inDialog	An ADM dialog.
inPluginRef	Plug-in reference.
inName	Internal identifier for the dialog. Must be unique. The suggested convention is to use a leading prefix of your company's name, as in MyCompanydialogxxxx .
inDialogID	Platform native resource ID. This ID refers to the platform-specific code provided to ADM to create the dialog on either the Macintosh or PC.

inInitProc	A callback with the following signature (see <code>ADMDialog.h</code>): ADMDialogInitProc (ADMDialogRef inDialog); Use to further initialize or customize the dialog. If special initialization is not required, pass NULL . Use inDialog to access the dialog items need initialization. If inInitProc returns an error, the dialog will not be created. See Appendix D for a list of error codes that can be returned.
inUserData	A 4-byte value that ADM keeps with the dialog; you can pass a pointer to a block of memory or some other value and retrieve it later using the sADMDialog->GetUserData() function.
inOptions	Currently unused. Always pass 0 (zero).

Returns

An ADM Item.

sADMDialog->CreateItem()

Create an item in a dialog

```
ADMItemRef ASAPI (*CreateItem)(ADMDialogRef inDialog, ASInt32
inItemID, ADMItemType inItemType, const ASRect* inBoundsRect,
ADMItemInitProc inInitProc, ADMUserData inUserData, ASInt32
inOptions);
```

Description

The **CreateItem()** function adds an ADM item to a dialog. To add an item, use the **kADMUniqueItemID** constant. Valid items types are defined in `ADMItem.h` and described in [Chapter 14, “The ADM Item Suite”](#).

ADM items in a dialog's item list resource are automatically added using this function when the [sADMDialog->Create\(\)](#) and [sADMDialog->Modal\(\)](#) functions are used.

Parameters

inDialog	An ADM dialog.
inItemID	Index of the item within inDialog 's list of items.
inItemType	Type of item to be created. Type: <code>ADMItemType</code> (see <code>ADMItem.h</code>)
inBoundsRect	Location and size of the item within the dialog's coordinate space. Type: ASRect (see <code>ASTypes.h</code>)

inInitProc	Initialization callback with the following signature (see <code>ADMItem.h</code>): ADMItemInitProc (ADMItemRef inItem); See Appendix D for a list of error codes that can be returned.
inUserData	A 4-byte value that ADM keeps with the item; you can pass a pointer to a block of memory or some other value and retrieve it later using the sADMItem->GetUserData() function.
inOptions	Currently unused. Always pass 0 (zero).

Returns

An ADM item.

See also

[sADMDialog->DestroyItem\(\)](#)
[sADMItem->GetUserData\(\)](#)

sADMDialog->CreateTimer()

Create a timer

```
ADMTimerRef ASAPI (*CreateTimer)(ADMDialogRef inDialog,
ASUInt32 inMilliseconds, ADMActionMask inAbortMask,
ADMDialogTimerProc inTimerProc, ADMDialogTimerAbortProc
inAbortProc, ASInt32 inOptions);
```

Description

The **CreateTimer()** function creates a timer for measuring time between events. Time is kept in milliseconds. User-supplied **inTimerProc** and **inAbortProc** determine actions that occur at end of the delay or if another action occurs. If the delay succeeds (i.e., not aborted) then **inTimerProc** is executed. If the action specified by the **inAbortMask** occurs, the **inAbortProc** is called. The possible values for **inAbortMask** are the same as the tracker masks and are defined in `ADMTracker.h`.

Parameters

inDialog	An ADM dialog.
inMilliseconds	Delay between events.
inAbortMask	Specifies actions that result in calling inAbortProc . Type: ADMActionMask (see <code>ASTypes.h</code>)

inTimerProc	Callback with the following signature (see <code>ADMDialog.h</code>): ASBoolean ADMDialogTimerProc (ADMDialogRef inDialog, ADMTimerRef inTimerID); This callback is executed at the end of an inMilliseconds delay. Returns a boolean. If it returns true , then inTimerProc will be called again after inMilliseconds . If it returns false then inTimerProc will no longer be called.
inAbortProc	Callback with the following signature (see <code>ADMDialog.h</code>): ADMDialogTimerAbortProc(ADMDialogRef inDialog, ADMTimerRef inTimerID, ADMAction inAbortAction); This callback is executed if an inAbortMask action occurs before the inMilliseconds delay completes.
inOptions	Currently unused. Always pass 0 (zero).

Returns

An ADM timer.

See also

[sADMDialog->AbortTimer\(\)](#)

sADMDialog->DefaultDraw()

Call ADM's default dialog draw function

```
void ASAPI (*DefaultDraw)(ADMDialogRef inDialog, ADMDrawerRef inDrawer);
```

Description

The **DefaultDraw()** function calls the dialog's current default draw function from within your custom dialog draw function. The arguments passed to the custom function are passed through to the **DefaultDraw()** call. Alternatively, you can set your own draw function using the [sADMDialog->SetDrawProc\(\)](#) function.

Parameters

inDialog	An ADM dialog.
inDrawer	An ADM drawer.

Returns

None.

See also

[sADMDialog->GetDrawProc\(\)](#)
[sADMDialog->SetDrawProc\(\)](#)

Example

```
void doNothingDrawHandler(ADMDialogRef dialog, ADMDrawerRef drawer) {
    sADMDialog->DefaultDraw(dialog, drawer);
}
```

sADMDialog->DefaultNotify()

Call ADM's default dialog notification function

```
void ASAPI (*DefaultNotify)(ADMDialogRef inDialog,
    ADMNotifierRef inNotifier);
```

Description

The **DefaultNotify()** function calls the dialog's current default notification function from within your custom dialog notification function. The arguments passed to the custom function are passed through to the **DefaultNotify()** call.

Parameters

inDialog	An ADM dialog.
inNotifier	An ADM notifier.

Returns

None.

Example

```
void doNothingNotificationHandler(ADMDialogRef dialog,
    ADMNotifierRef notifier) {
    sADMDialog->DefaultNotify(dialog, notifier);
}
```

sADMDialog->DefaultTrack()

Call ADM's default dialog tracker function

```
ASBoolean ASAPI (*DefaultTrack)(ADMDialogRef inDialog,
    ADMTrackerRef inTracker);
```

Description

The **DefaultTrack()** function calls the dialog's current default tracker function from within your custom dialog tracker function. The arguments passed to the custom function are passed through to the **DefaultTrack()** call.

Parameters

inDialog	An ADM dialog.
inTracker	An ADM tracker.

Returns

true if tracker successfully called; **false** otherwise.

Example

```
ASBoolean doNothingTrackHandler(ADMDialogRef dialog, ADMTrackerRef
tracker) {
    bool rc = sADMDialog->DefaultTrack(dialog, tracker);
    if rc {
        //the default tracker is successfully called
    }
    else
    {
        //problem with execution of default tracker
    }

    return rc;
}
```

sADMDialog->Destroy()

Remove an ADM modeless dialog from memory

```
void ASAPI (*Destroy)(ADMDialogRef inDialog);
```

Description

The **Destroy()** function removes an ADM non-modal dialog and its resources from memory. Unlike modal dialogs, non-modal dialogs must be destroyed when your plug-in finishes with them. If your plug-in shuts down before ADM shuts down, you should call **Destroy()** first. But if ADM shuts down first, it will automatically call **Destroy()** on all dialogs before shutting down. If you have used [sADMDialog->SetDestroyProc\(\)](#) to give the dialog a custom destroy function, your function will be triggered by this **Destroy()**.

NOTE: It is imperative that you do NOT call **Destroy()** twice. Therefore, you should keep a reference of whether or not your **Destroy()** callback has been called so you can check it before calling **Destroy()** again.

You will likely call **Destroy()** in response to a PICA shutdown message or an ADM close window notification.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

None.

See also

[sADMDialog->Create\(\)](#)
[sADMDialog->SetDestroyProc\(\)](#)

sADMDialog->DestroyItem()

Remove an item from a dialog

```
void ASAPI (*DestroyItem)(ADMDialogRef inDialog, ADMItemRef
inItem);
```

Description

The **DestroyItem()** function removes **inItem** from **inDialog**. This function causes the **inItem**'s **Destroy()** function to be called, freeing acquired resources.

Items in an ADM dialog are automatically destroyed using this function when the [sADMDialog->Destroy\(\)](#) function is called and when a modal dialog is terminated by the user.

Parameters

inDialog	An ADM dialog.
inItem	An ADM item.

Returns

None.

See also

[sADMDialog->CreateItem\(\)](#)
[sADMItem->Destroy\(\)](#)

sADMDialog->DisplayAsModal()

Make dialog a modal dialog and disable all floating palettes

```
ASInt32 ASAPI (*DisplayAsModal)(ADMDialogRef inDialog);
```

Description

The **DisplayAsModal()** function forces an ADM dialog to be displayed and behave as a modal dialog. This API disables all ADM non-modal dialogs. It only works for ADM modal dialogs.

How is this API used? If you have a complex dialog that is used often in your application and is expensive to create, then this API can be useful. Create the dialog

once and use this API to display it over and over again in your application. You save the overhead of recreating it from scratch each time it is displayed. [sADMDialog->DisplayAsPopupModal\(\)](#) has the same utility.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

ID of the ADM item used to use to dismiss the dialog.

See also

[sADMDialog->DisplayAsPopupModal\(\)](#)

sADMDialog->DisplayAsPopupModal()

Display the pop-up dialog provided

```
ASInt32 ASAPI (*DisplayAsPopupModal)(ADMDialogRef inDialog);
```

Description

The **DisplayAsPopupModal()** function displays the pop-up dialog **inDialog**—it does not create or destroy the dialog.

A traditional modal dialog can only be dismissed by either the **OK** or **Cancel** button. The popup modal is similar to a modal dialog in that it only stays up while the end-user is using it. However the popup modal has no **OK/Cancel** button, but is dismissed when the end user makes a selection, hits the escape button, etc.

How is this API used? If you have a complex dialog that is used often in your application and is expensive to create, then this API can be useful. Create the dialog once and use this API to display it over and over again in your application. You save the overhead of recreating it from scratch each time it is displayed. [sADMDialog->DisplayAsModal\(\)](#) has the same utility.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

ID of the ADM item used to use to dismiss the dialog.

See also

[sADMDialog->DisplayAsModal\(\)](#)

sADMDialog->Enable()

Enable or disable a dialog

```
void ASAPI (*Enable)(ADMDialogRef inDialog, ASBoolean
inEnable);
```

Description

The **Enable()** function enables or disables a dialog. ADM automatically disables any enabled dialogs when a modal dialog is created. When the modal dialog is destroyed, other ADM dialogs are returned to their previous enabled state. A disabled ADM dialog is dimmed and unusable.

Parameters

inDialog	An ADM dialog.
inEnable	Pass true to enable the dialog and false to disable it.

Returns

None.

See also

[sADMDialog->IsEnabled\(\)](#)

sADMDialog->EndModal()

Dismiss a modal dialog

```
ASBoolean ASAPI (*EndModal)(ADMDialogRef inDialog, ASInt32
inModalResultID, ASBoolean inCancelling);
```

Description

The **EndModal()** function is used within a modal dialog handler to dismiss the dialog. This function could be used, for instance, to implement shortcut selections to close the dialog—for example, allow the user to select the dialog and press the **Option** key, resulting in the handler immediately performing the action and automatically dismissing the dialog. **EndModal()** will likely be called in response to a given item being selected. The handler calling this function can do so at either the dialog or the item level. For instance, a button item that is used to accept a password might automatically call **EndModal()** if the wrong password were entered.

NOTE: Do not use this function in your initialization procedure. Instead, when an error occurs during initialization, the Init proc should return something other than **kNoErr**, and the dialog won't start up or show up.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

inModalResultID	ID of the item used to dismiss inDialog . (Returned by the sADMDialog->Modal() function.)
inCancelling	true if the dialog options are to be accepted (OK); false if inDialog is to be cancelled without changes being made (Cancel).

Returns

true if **inDialog** is successfully dismissed; **false** otherwise. If **false** is returned, then some internal error has occurred.

sADMDialog->GetBoundsRect()

Get the absolute position and size of a dialog

```
void ASAPI (*GetBoundsRect)(ADMDialogRef inDialog, ASRect*  
outBoundsRect);
```

Description

The `GetBoundsRect()` function returns the current size and position of **inDialog** in screen coordinates.

Parameters

inDialog	An ADM dialog.
outBoundsRect	The current size and position of inDialog in screen coordinates. Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDialog->SetBoundsRect\(\)](#)
[sADMDialog->Move\(\)](#)
[sADMDialog->GetLocalRect\(\)](#)
[sADMDialog->SetLocalRect\(\)](#)
[sADMDialog->Size\(\)](#)

sADMDialog->GetCancelItemID()

Set the item that cancels a dialog

```
ASInt32 ASAPI (*GetCancelItemID)(ADMDialogRef inDialog);
```

Description

The **GetCancelItemID()** function returns the ID of the **Cancel** item in **inDialog**. The **Cancel** item is notified when the **Esc** key is pressed. On the Macintosh platform, it is also notified when **Command** . is pressed.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

ID of the **Cancel** item in **inDialog**.

See also

[sADMDialog->SetCancelItemID\(\)](#)

sADMDialog->GetCursorID()

Get the dialog's cursor ID

```
void ASAPI (*GetCursorID)(ADMDialogRef inDialog, SPPluginRef*
outPluginRef, ASInt32* outCursorID);
```

Description

The **GetCursorID()** function returns the resource ID of the cursor to be displayed when the mouse position is inside the dialog.

Parameters

inDialog	An ADM dialog.
outPluginRef	Plug-in reference.
outCursorID	Resource ID of the cursor to be displayed when the mouse position is inside the dialog. (See <code>ADMResource.h</code> for a list of built-in ADM cursor IDs.)

Returns

None.

See also

[sADMDialog->SetCursorID\(\)](#)

sADMDialog->GetDefaultItemID()

Get the default item of a dialog

```
ASInt32 ASAPI (*GetDefaultItemID)(ADMDialogRef inDialog);
```


Description

The **GetDefaultItemID()** function returns the ID of the item in **inDialog** that is the default item. The default item is notified when the **Enter** key is pressed.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

ID of the item in **inDialog** that is the default item.

See also

[sADMDialog->SetDefaultItemID\(\)](#)

sADMDialog->GetDestroyProc()

Get the ADM destroy function being used for the dialog

```
ADMDialogDestroyProc ASAPI (*GetDestroyProc)(ADMDialogRef  
inDialog);
```

Description

The **GetDestroyProc()** function gets the Destroy proc being used for **inDialog**. Because ADM calls the dialog's destroy function when it is deleted, you should not call the returned function directly.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

The Destroy proc. If you have not called [sADMDialog->SetDestroyProc\(\)](#), returns **NULL** (not the default Destroy proc). Type: **ADMDialogDestroyProc** (see **ADMDialog.h**)

See also

[sADMDialog->SetDestroyProc\(\)](#)

sADMDialog->GetDialogName()

Get the name of a dialog

```
ASAPI const char* (*GetDialogName)(ADMDialogRef inDialog);
```

Description

The **GetDialogName()** function gets a reference to the unique character ID name of **inDialog**. This is the name specified in the [sADMDialog->Create\(\)](#) and [sADMDialog->Modal\(\)](#) functions; it can also be set using [sADMDialog->SetDialogName\(\)](#).

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

Reference to the unique character ID name of **inDialog**

See also

[sADMDialog->SetDialogName\(\)](#)
[sADMDialogGroup->GetNamedDialog\(\)](#)

sADMDialog->GetDialogStyle()

Get the style of a dialog

```
ADMDialogStyle WINAPI (*GetDialogStyle)(ADMDialogRef inDialog);
```

Description

The **GetDialogStyle()** function gets the current style of an ADM Dialog as discussed in [Dialog Basics: Styles](#), and the [sADMDialog->Create\(\)](#) and [sADMDialog->Modal\(\)](#) functions.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

The current style of **inDialog**. Type: **ADMDialogStyle** (see `ADMBasic.h`)

See also

[sADMDialog->SetDialogStyle\(\)](#)
[sADMDialog->Create\(\)](#)
[sADMDialog->Modal\(\)](#)

sADMDialog->GetDrawProc()

Get the ADM drawing function being used for the dialog

```
ADMDialogDrawProc WINAPI (*GetDrawProc)(ADMDialogRef inDialog);
```

Description

The **GetDrawProc()** function gets the Drawer proc being used for **inDialog**. However, rather than getting and calling a dialog's draw function in this fashion, you are more likely to use the [sADMDialog->DefaultDraw\(\)](#) function.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

The Drawer proc being used for **inDialog**. If you have not called [sADMDialog->SetDrawProc\(\)](#), returns **NULL** (not the default Drawer proc). Type: **ADMDialogDrawProc** (see **ADMDialog.h**)

See also

[sADMDialog->SetDrawProc\(\)](#)
[sADMDialog->DefaultDraw\(\)](#)

sADMDialog->GetFont()

Get the Font style

```
ADMFont ASAPI (*GetFont)(ADMDialogRef inDialog);
```

Description

The **GetFont()** function retrieves **inDialog**'s font style. Typically you would use this information for spacing requirements.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

inDialog's font style. Type: **ADMFont** (see **ADMTypes.h**)

See also

[sADMDialog->SetFont\(\)](#)

sADMDialog->GetHorizontalIncrement()

Get the horizontal increment to use when resizing a window

```
ASInt32 ASAPI (*GetHorizontalIncrement)(ADMDialogRef  
inDialog);
```

Description

The **GetHorizontalIncrement()** function returns the increment to use when resizing **inDialog** horizontally. The increment is the amount that will be added to or subtracted from the dialog's width as the user increases or decreases its size.

By default, a dialog's horizontal increment is **1**, making it sizable to any given width. The dialog's width will always be its minimum width plus a multiple of the horizontal increment.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

Increment to use when resizing **inDialog** horizontally.

See also

[sADMDialog->SetHorizontalIncrement\(\)](#)
[sADMDialog->GetVerticalIncrement\(\)](#)
[sADMDialog->SetVerticalIncrement\(\)](#)

sADMDialog->GetID()

Get the ID of a dialog

```
ASInt32 WINAPI (*GetID)(ADMDialogRef inDialog);
```

Description

The **GetID()** function gets the ID of **inDialog**. This is the ID of the resource used to create it and was specified when the dialog was created with [sADMDialog->Create\(\)](#) or [sADMDialog->Modal\(\)](#).

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

The dialog ID of **inDialog**.

See also

[sADMDialog->Create\(\)](#)
[sADMDialog->Modal\(\)](#)

sADMDialog->GetItem()

Get an item of a dialog

```
ADMItemRef ASAPI (*GetItem)(ADMDialogRef inDialog, ASInt32  
inItemID);
```

Description

The **GetItem()** function gets a reference to an item in **inDialog**. Once you have the item reference, you can use the ADM Item suite functions to access it. To access all the items of a dialog, get item **kADMFirstItemID** and then use the [sADMDialog->GetNextItem\(\)](#) function to walk the dialog item list.

Parameters

inDialog	An ADM dialog.
inItemID	An ADM item ID. Can be a known item ID or it can be one of ADMs standard ADMStandardDialogItemID constants (see ADMDialog.h).

Returns

An ADM item in **inDialog**.

See also

[sADMDialog->GetNextItem\(\)](#)

sADMDialog->GetLocalRect()

Get the size of a dialog

```
void ASAPI (*GetLocalRect)(ADMDialogRef inDialog, ASRect*  
outLocalRect);
```

Description

The **GetLocalRect()** function gets the size of the **inDialog** window in (0,0)-based coordinates. The **bottom** and **right** members of the **ASRect** structure (see **ASTypes.h**) are the dialog's size. In Windows parlance, this function returns the client area—the area accessible to the client. It does not include the title bar or window frame.

Parameters

inDialog	An ADM dialog.
outLocalRect	The inDialog window in (0,0)-based coordinates. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDialog->SetLocalRect\(\)](#)

sADMDialog->GetMask()

Get mask used for tracker function

```
ADMActionMask WINAPI (*GetMask)(ADMDialogRef inDialog);
```

Description

The **GetMask()** function gets the mask used for controlling which events are received by the tracker.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

The mask. Type: **ADMActionMask** (see `ADMTypes.h`)

See also

[sADMDialog->SetMask\(\)](#)

sADMDialog->GetMaxHeight()

Get the maximum height of a dialog's window

```
ASInt32 WINAPI (*GetMaxHeight)(ADMDialogRef inDialog);
```

Description

The **GetMaxHeight()** function returns the maximum height in pixels of resizable dialog **inDialog**. When the user is resizing the dialog, ADM will not allow its height to be greater than this value.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

Maximum height (in pixels) of **inDialog**.

See also

[sADMDialog->SetMaxHeight\(\)](#)
[sADMDialog->GetMinHeight\(\)](#)

sADMDialog->SetMinHeight()
sADMDialog->GetMaxWidth()
sADMDialog->SetMaxWidth()
sADMDialog->GetMinWidth()
sADMDialog->SetMinWidth()

sADMDialog->GetMaxWidth()

Get the maximum width of a dialog's window

```
ASInt32 WINAPI (*GetMaxWidth)(ADMDialogRef inDialog);
```

Description

The **GetMaxWidth()** function returns the maximum width in pixels of resizable dialog **inDialog**. When the user is resizing the dialog, ADM will not allow its width to be greater than this value.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

Maximum width (in pixels) of **inDialog**.

See also

sADMDialog->SetMaxWidth()
sADMDialog->GetMaxHeight()
sADMDialog->SetMaxHeight()
sADMDialog->GetMinWidth()
sADMDialog->SetMinWidth()
sADMDialog->GetMinHeight()
sADMDialog->SetMinHeight()

sADMDialog->GetMinHeight()

Get the minimum height of a dialog's window

```
ASInt32 WINAPI (*GetMinHeight)(ADMDialogRef inDialog);
```

Description

The **GetMinHeight()** function returns the minimum height in pixels of resizable dialog **inDialog**. When the user is resizing the dialog, ADM will not allow its height to be less than this value.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

Minimum height (in pixels) of **inDialog**.

See also

[sADMDialog->SetMinHeight\(\)](#)
[sADMDialog->GetMaxHeight\(\)](#)
[sADMDialog->SetMaxHeight\(\)](#)
[sADMDialog->GetMaxWidth\(\)](#)
[sADMDialog->SetMaxWidth\(\)](#)
[sADMDialog->GetMinWidth\(\)](#)
[sADMDialog->SetMinWidth\(\)](#)

sADMDialog->GetMinWidth()

Set the minimum width of a dialog's window

```
ASInt32 ASAPI (*GetMinWidth)(ADMDialogRef inDialog);
```

Description

The **GetMinWidth()** function returns the minimum width in pixels of resizable dialog **inDialog**. When the user is resizing the dialog, ADM will not allow its width to be less than this value.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

Minimum width (in pixels) of **inDialog**.

See also

[sADMDialog->SetMinWidth\(\)](#)
[sADMDialog->GetMaxWidth\(\)](#)
[sADMDialog->SetMaxWidth\(\)](#)
[sADMDialog->GetMinHeight\(\)](#)
[sADMDialog->SetMinHeight\(\)](#)
[sADMDialog->GetMaxHeight\(\)](#)
[sADMDialog->SetMaxHeight\(\)](#)

sADMDialog->GetNextItem()

Get the item following another

```
ADMItemRef ASAPI (*GetNextItem)(ADMDialogRef inDialog,  
ADMItemRef inItem);
```


Description

The **GetNextItem()** function is used to iterate through **inDialog**'s list of items. When **inItem** is the last in the list, **NULL** is returned.

To access all of the items of **inDialog**, use the [sADMDialog->GetItem\(\)](#) function with item ID **kADMFirstItemID** to get an **inItem**. Then use **GetNextItem()** to walk through the dialog item list.

Parameters

inDialog	An ADM dialog.
inItem	The ADM item whose following item in inDialog is to be returned.

Returns

The next ADM item in **inDialog**; if **inItem** is the last item, **NULL** is returned.

See also

[sADMDialog->GetPreviousItem\(\)](#)
[sADMDialog->GetItem\(\)](#)

sADMDialog->GetNotifierData()

Get notification data

```
ADMUserData ASAPI (*GetNotifierData)(ADMDialogRef inDialog);
```

Description

The **GetNotifierData()** function gets the notification data of **inDialog**, if any.

Parameters

inDialog	An ADM item.
-----------------	--------------

Returns

The notifier data. Type: **ADMUserData** (see `ADMTypes.h`)

See also

[sADMDialog->SetNotifierData\(\)](#)

sADMDialog->GetNotifyProc()

Get the ADM notification function being used for the dialog

```
ADMDialogNotifyProc ASAPI (*GetNotifyProc)(ADMDialogRef  
inDialog);
```

Description

The **GetNotifyProc()** function gets the notification function being used for **inDialog**. Rather than getting and calling a dialog's notification function directly, you are more likely to use the [sADMDialog->DefaultNotify\(\)](#) function.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

The notification function being used for **inDialog**. If you have not called [sADMDialog->SetNotifyProc\(\)](#), returns **NULL** (not the default Notify proc). Type: **ADMDialogNotifyProc** (see **ADMDialog.h**)

See also

[sADMDialog->SetNotifyProc\(\)](#)
[sADMDialog->DefaultNotify\(\)](#)

sADMDialog->GetPluginRef()

Get the plug-in that created a dialog

```
SPPluginRef ASAPI (*GetPluginRef)(ADMDialogRef inDialog);
```

Description

The **GetPluginRef()** function returns the **SPPluginRef** of the plug-in that added **inDialog**. A plug-in reference may be useful to, for example, send the plug-in a message.

See the *Adobe PICA Programmer's Guide and Reference* for more information on directly interfacing with a plug-in.

Parameters

inDialog	AN ADM dialog.
-----------------	----------------

Returns

The **SPPluginRef** (see **SPPlugs.h**) of the plug-in that added **inDialog**.

sADMDialog->GetPreviousItem()

Get the item previous to another

```
ADMItemRef ASAPI (*GetPreviousItem)(ADMDialogRef inDialog,  
ADMItemRef inItem);
```

Description

The **GetPreviousItem()** function iterates backwards through **inDialog**'s list of items. When **inItem** is first in the list, **NULL** is returned.

To access all of the items of **inDialog**, use the [sADMDialog->GetItem\(\)](#) function with item ID **kADMFirstItemID** to get an **inItem**. Then use **GetPreviousItem()** to walk through the dialog item list.

Parameters

inDialog	An ADM dialog.
inItem	The ADM item whose previous item in inDialog is to be returned.

Returns

The previous ADM item in **inDialog**; if the **inItem** is the first item, **NULL** is returned.

See also

[sADMDialog->GetNextItem\(\)](#)
[sADMDialog->GetItem\(\)](#)

sADMDialog->GetText()

Get the dialog title

```
void ASAPI (*GetText)(ADMDialogRef inDialog, char* outText,  
ASInt32 inMaxLength);
```

Description

The **GetText()** function retrieves **inDialog**'s text into buffer **outText**. A dialog's text property is its title, which is displayed in its window title bar.

Parameters

inDialog	An ADM dialog.
outText	Buffer into which inDialog 's text property is placed
inMaxLength	Size of outText .

Returns

None.

See also

[sADMDialog->SetText\(\)](#)
[sADMDialog->GetTextLength\(\)](#)

sADMDialog->GetTextLength()

Get the length of dialog's title

```
ASInt32 ASAPI (*GetTextLength)(ADMDialogRef inDialog);
```

Description

The **GetTextLength()** function gets the number of characters **inDialog**'s text. A dialog's text property is its title, which is displayed in its window title bar.

Parameters

inDialog

Returns

Number of characters **inDialog**'s text.

See also

[sADMDialog->SetText\(\)](#)

[sADMDialog->GetText\(\)](#)

sADMDialog->GetTrackProc()

Get the ADM tracker function being used for the dialog

```
ADMDialogTrackProc ASAPI (*GetTrackProc)(ADMDialogRef  
inDialog);
```

Description

The **GetTrackProc()** function gets the event tracking proc being used for **inDialog**. Rather than getting and calling a dialog's tracker function directly, you are more likely to use the [sADMDialog->DefaultTrack\(\)](#) function.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

The event tracking proc being used for **inDialog**. If you have not called [sADMDialog->SetTrackProc\(\)](#), returns **NULL** (not the default track proc). Type: **ADMDialogTrackProc** (see **ADMDialog.h**)

See also

[sADMDialog->SetTrackProc\(\)](#)

[sADMDialog->DefaultTrack\(\)](#)

sADMDialog->GetUserData()

Get the user data pointer for a dialog

```
ADMUserData ASAPI (*GetUserData)(ADMDialogRef inDialog);
```

Description

The **GetUserData()** function returns the 4-byte user value stored with **inDialog**. It is initialized by the ADM Dialog suite's **Create()** and **Modal()** functions. You can also set it with [sADMDialog->SetUserData\(\)](#).

The meaning of the value is defined by the dialog's creator. It is likely a pointer to a data structure—for instance, the plug-in's globals. For some dialogs, it might be a simple 4-byte type, such as a **long** or a **fixed** number.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

A 4-byte value that ADM keeps with **inDialog**. Type: **ADMUserData** (see **ADMTypes.h**)

See also

[sADMDialog->SetUserData\(\)](#)

sADMDialog->GetVerticalIncrement()

Get the vertical increment to use when resizing a window

```
ASInt32 ASAPI (*GetVerticalIncrement)(ADMDialogRef inDialog);
```

Description

The **GetVerticalIncrement()** function returns the increment to use when resizing **inDialog** vertically. The increment is the amount that will be added to or subtracted from the dialog's height as the user increases or decreases its size.

By default, a dialog's vertical increment is **1**, making it sizable to any given height. The dialog's width will always be its minimum height plus a multiple of the vertical increment.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

Increment to use when resizing **inDialog** vertically.

See also

[sADMDialog->SetVerticalIncrement\(\)](#)
[sADMDialog->GetHorizontalIncrement\(\)](#)
[sADMDialog->SetHorizontalIncrement\(\)](#)

sADMDialog->GetWindowRef()

Get the window of a dialog

```
ASWindowRef ASAPI (*GetWindowRef)(ADMDialogRef inDialog);
```

Description

The **GetWindowRef()** function gets the platform window reference for **inDialog**. This reference might be used to draw directly into the window.

On Macintosh, this is the window's **GrafPort**:

```
typedef struct GrafPort *ASWindowRef;
```

On Windows, this is the same as a Windows **HWND**:

```
typedef void * ASWindowRef;
```

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

Platform window reference for **inDialog**. Type: **ASWindowRef** (see **ASTypes.h**)

See also

[sADMDialog->SetWindowRef\(\)](#)

sADMDialog->Invalidate()

Invalidate the window of a dialog

```
void ASAPI (*Invalidate)(ADMDialogRef inDialog);
```

Description

The **Invalidate()** function invalidates **inDialog**'s window, causing it to be redrawn the next time the screen is updated.

NOTE: On Windows, this call does not erase the background when repainting. Thus, if you use [sADMDialog->SetDrawProc\(\)](#) to set up a custom background, there is a problem. You must use the Windows Invalidate calls with the **bErase** flag set to **true** in order to force the background to repaint.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

None.

See also

[sADMDialog->InvalidateRect\(\)](#)

sADMDialog->InvalidateRect()

Invalidate the specified rectangle

```
void WINAPI (*InvalidateRect)(ADMDialogRef inDialog, const
ASRect* inInvalRect);
```

Description

The **InvalidateRect()** function invalidates the specified rectangle, causing it to be redrawn next time the screen is updated.

NOTE: On Windows, this call does not erase the background when repainting. Thus, if you use [sADMDialog->SetDrawProc\(\)](#) to set up a custom background, there is a problem. You must use the Windows Invalidate calls with the **bErase** flag set to **true** in order to force the background to repaint.

Parameters

inDialog	An ADM dialog.
inInvalRect	A rectangle within inDialog . Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDialog->Invalidate\(\)](#)

sADMDialog->IsActive()

Get whether or not a dialog is active

```
ASBoolean WINAPI (*IsActive)(ADMDialogRef inDialog);
```

Description

The **IsActive()** function determines whether a floating dialog is currently active. To change its state, use the [sADMDialog->Activate\(\)](#) function.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

true if **inDialog** is currently active; **false** otherwise.

See also

[sADMDialog->Activate\(\)](#)

sADMDialog->IsCollapsed()

Determines whether a dialog palette is collapsed

```
ASBoolean ASAPI (*IsCollapsed)(ADMDialogRef inDialog);
```

Description

The **IsCollapsed()** function determines whether a specific non-modal dialog palette is collapsed.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

true if **inDialog** is collapsed; **false** otherwise.

sADMDialog->IsDialogContextHidden()

Get whether or not a dialog's context is hidden

```
ASBoolean ASAPI (*IsDialogContextHidden)(ADMDialogRef  
inDialog);
```

Description

The **IsDialogContextHidden()** function returns a boolean indicating whether **inDialog**'s context is hidden.

Context here refers to the context involved in a context switch between different applications or processes by the OS. This call does not say anything about the visibility of the dialog. If the application is not the active application, then **IsDialogContextHidden** returns **true**. If an application is active, then all its dialogs will return **false** for **IsDialogContextHidden**.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

true if **inDialog**'s context is hidden; **false** otherwise.

sADMDialog->IsEnabled()

Get whether or not a dialog is enabled

```
ASBoolean ASAPI (*IsEnabled)(ADMDialogRef inDialog);
```

Description

The **IsEnabled()** function determines whether **inDialog** is currently enabled. To change its state, use the [sADMDialog->Enable\(\)](#) function.

A disabled ADM dialog is dimmed and unusable. ADM automatically disables any enabled dialogs when a modal dialog is created. When the modal dialog is destroyed, other ADM dialogs are returned to their previously enabled state.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

true if **inDialog** is currently enabled; **false** otherwise.

See also

[sADMDialog->Enable\(\)](#)

sADMDialog->IsForcedOnScreen()

Check whether the dialog force-on-screen flag is set

```
ASBoolean ASAPI (*IsForcedOnScreen)(ADMDialogRef inDialog);
```

Description

The **IsForcedOnScreen()** function checks whether the dialog force-on-screen flag for **inDialog** is set.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

true if force-on-screen flag is set; **false** otherwise.

See also

[sADMDialog->SetForcedOnScreen\(\)](#)

sADMDialog->IsUpdateEnabled()

Check whether the update enabled flag for this dialog is set

```
ASBoolean ASAPI (*IsUpdateEnabled)(ADMDialogRef inDialog);
```

Description

The **IsUpdateEnabled()** function checks whether the update enabled flag for **inDialog** is set. If this function returns **false**, then no screen update will occur—that is, a call to [sADMDialog->Update\(\)](#) does nothing.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

true if the update enabled flag is set; **false** otherwise.

See also

[sADMDialog->SetUpdateEnabled\(\)](#)
[sADMDialog->Update\(\)](#)

sADMDialog->IsVisible()

Get whether or not a dialog is visible

```
ASBoolean ASAPI (*IsVisible)(ADMDialogRef inDialog);
```

Description

The **IsVisible()** function determines if **inDialog** is currently visible. To change its state, use the [sADMDialog->Show\(\)](#) function.

If the dialog is part of a tabbed group but is not the front-most tab, this function returns **false**. Note that in this case, this value should not be used to restore a saved state.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

true if **inDialog** is currently visible; **false** otherwise. If **inDialog** is part of a tabbed group but is not the front-most tab, returns **false**.

See also

[sADMDialog->Show\(\)](#)

sADMDialog->LoadToolTips()

Set the tool tips for a dialog

```
void ASAPI (*LoadToolTips)(ADMDialogRef inDialog, ASInt32  
inStringID);
```

Description

The **LoadToolTips()** function defines the tool tip strings for **inDialog**. The **inStringID** is the base string resource for the dialog. Strings in the Macintosh string list (**'STR#'**) and Windows string resources (**stringID + (1 to n)**) are read and assigned to the dialog items.

Parameters

inDialog	An ADM dialog.
inStringID	Base string resource for inDialog .

Returns

None.

sADMDialog->LocalToScreenPoint()

Convert a dialog point to screen coordinates

```
void ASAPI (*LocalToScreenPoint)(ADMDialogRef inDialog,  
ASPoint* ioPoint);
```

Description

The **LocalToScreenPoint()** function converts a point in **inDialog** to its screen coordinates.

Parameters

inDialog	An ADM dialog.
ioPoint	The point. Type: ASPoint (see ASTypes.h)

Returns

None.

See also

[sADMDialog->ScreenToLocalPoint\(\)](#)

sADMDialog->LocalToScreenRect()

Convert a dialog rectangle to screen coordinates

```
void ASAPI (*LocalToScreenRect)(ADMDialogRef inDialog, ASRect*
ioRect);
```

Description

The **LocalToScreenRect ()** function converts a rectangle in **inDialog** to its equivalent screen coordinates.

Parameters

inDialog	An ADM dialog.
ioRect	The rect. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDialog->ScreenToLocalRect\(\)](#)

sADMDialog->Modal()

Create a new ADM modal dialog

```
ASInt32 ASAPI (*Modal)(SPPluginRef inPluginRef, const char*
inName, ASInt32 inDialogID, ADMDialogStyle inDialogStyle,
ADMDialogInitProc inInitProc, ADMUserData inUserData, ASInt32
inOptions);
```

Description

The **Modal ()** function creates a modal dialog. Modal dialogs typically display some information and require the interaction of the user, then disappear when the user clicks the **Okay** or **Cancel** button. The function returns an **ASInt32** which is the ID of the item that terminates the dialog. Typically this will be the **Okay** or **Cancel** button, but it can be any item in the dialog. (See the [sADMDialog->EndModal\(\)](#) function description.)

The **inPluginRef** is the plug-in creating the dialog. Use the reference passed in the **SPMessageData** data structure when the plug-in is loaded. **inName** is the internal identifier for the dialog. It is the only ID that you can guarantee is unique, assuming that you use a unique name. **inDialogID** is the platform native resource ID to use in creating the dialog. ADM reads the resource correctly and creates the window and any dialog items. **inDialogStyle** indicates the type of dialog; there are only a few valid styles, as described in [Dialog Basics: Styles](#). **inInitProc** is called after ADM has created the dialog; it provides the you with the opportunity to further initialize or customize the dialog. **inUserData** is a 4-byte value that ADM keeps with the dialog. You can pass a pointer to a block of memory or some other value and retrieve it later using the [sADMDialog->GetUserData\(\)](#) function.

ADM automatically destroys the dialog for you when it is dismissed by the user. The return value is the ID of the item used to dismiss the dialog.

Parameters

inPluginRef	Plug-in reference.
inName	Internal identifier for the dialog. Must be unique. The suggested convention is to use a leading prefix of your company's name, as in MyCompanydialogxxxx .
inDialogID	Platform native resource ID. This ID refers to the platform-specific code provided to ADM to create the dialog on either the Macintosh or PC.
inDialogStyle	Style of dialog. See Dialog Basics: Styles . Type: ADMDialogStyle (see ADMDialog.h)
inInitProc	A callback with the following signature (see ADMDialog.h): ADMDialogInitProc (ADMDialogRef inDialog); Use to further initialize or customize the dialog. If special initialization is not required, pass NULL . Use inDialog to access the dialog items that need initialization. If inInitProc returns an error, the dialog will not be created: do not use sADMDialog->EndModal() if there is an error. See Appendix D for a list of error codes that can be returned.
inUserData	A 4-byte value that ADM keeps with the dialog; you can pass a pointer to a block of memory or some other value and retrieve it later using the sADMDialog->GetUserData() function. Type: ADMUserData (see ADMTypes.h)
inOptions	Special options. If no special options are required, pass 0 (zero). See Possible values for inOptions .

Returns

ID of the item used to dismiss the dialog.

See also

[sADMDialog->Create\(\)](#)
[sADMDialog->EndModal\(\)](#)
[sADMDialog->GetUserData\(\)](#)

sADMDialog->Move()

Set the relative position of a dialog

```
void ASAPI (*Move)(ADMDialogRef inDialog, ASInt32
inHorizPosition, ASInt32 inVertPosition);
```

Description

The **Move()** function moves **inDialog**'s position to the screen coordinates specified by **inHorizPosition** and **inVertPosition**. Therefore, **Move(myDialog, 0, 0)** moves **myDialog** to the upper left of the screen. If this function is used to move **inDialog** off-screen, **inDialog** will not be forced back onto the screen.

Parameters

inDialog	An ADM dialog.
inHorizPosition	New horizontal screen coordinate for inDialog .
inVertPosition	New vertical screen coordinate for inDialog .

Returns

None.

sADMDialog->PopupModal()

Pop up a modal dialog box

```
ASInt32 ASAPI (*PopupModal)(SPPluginRef inPluginRef, const
char* inName, ASInt32 inDialogID, ADMDialogInitProc
inInitProc, ADMUserData inUserData, ASInt32 inOptions);
```

Description

The **PopupModal()** function pops up a modal dialog box. The style is **kADMPopupControlDialogStyle**. There is no other choice for style.

Parameters

inPluginRef	Plug-in reference.
--------------------	--------------------

inName	Internal identifier for the dialog. Must be unique. The suggested convention is to use a leading prefix of your company's name, as in MyCompanydialogxxxx .
inDialogID	Platform native resource ID. This ID refers to the platform-specific code provided to ADM to create the dialog on either the Macintosh or PC.
inInitProc	A callback with the following signature (see <code>ADMDialog.h</code>): ADMDialogInitProc (ADMDialogRef inDialog); Use to further initialize or customize the dialog. If special initialization is not required, pass NULL . Use inDialog to access the dialog items that need initialization. If inInitProc returns an error, the dialog will not be created: do not use sADMDialog->EndModal() if there is an error. See Appendix D for a list of error codes that can be returned.
inUserData	A 4-byte value that ADM keeps with the dialog; you can pass a pointer to a block of memory or some other value and retrieve it later using the sADMDialog->GetUserData() function. Type: ADMUserData (see <code>ADMTypes.h</code>)
inOptions	Currently unused. Always pass 0 (zero).

Returns

ID of the item used to dismiss the dialog.

See also

[sADMDialog->Modal\(\)](#)

sADMDialog->RegisterItemType()

Add a custom item type

```
ASErr ASAPI (*RegisterItemType)(SPPluginRef inPluginRef,
    ADMItemType inItemType);
```

Description

NOTE: This API is deprecated in ADM V2.8. Custom item types will no longer be supported.

The **RegisterItemType()** function makes a custom item type available. Once registered, the plug-in should handle the **ADMCreateCustomItem** plug-in message. See [Custom Item Types](#) in [Chapter 1, "ADM Overview"](#) for more information.

Parameters

inPluginRef	Plug-in reference.
inItemType	The custom item type. Type: ADMItemType (see <code>ADMTypes.h</code>)

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. Possible errors include **kADMCustomResourceExistsError**, **kBadParameterErr**, **kOutOfMemoryError**. See [Appendix D](#) for a list of ADM error codes.

See also

[sADMDialog->UnregisterItemType\(\)](#)

sADMDialog->ScreenToLocalPoint()

Convert a screen point to dialog coordinates

```
void ASAPI (*ScreenToLocalPoint)(ADMDialogRef inDialog,
ASPoint* ioPoint);
```

Description

The **ScreenToLocalPoint()** function converts a point on the screen to a point relative to the dialog coordinates.

Parameters

inDialog	An ADM dialog.
ioPoint	The point. Type: ASPoint (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDialog->LocalToScreenPoint\(\)](#)

sADMDialog->ScreenToLocalRect()

Convert a screen rectangle to dialog coordinates

```
void ASAPI (*ScreenToLocalRect)(ADMDialogRef inDialog, ASRect*
ioRect);
```


Description

The **ScreenToLocalRect ()** function converts a rectangle in screen coordinates to a rectangle in the coordinate space of **inDialog**.

Parameters

inDialog	An ADM dialog.
ioRect	The rect. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDialog->LocalToScreenRect\(\)](#)

sADMDialog->SendNotify()

Send a notification to a dialog

```
void ASAPI (*SendNotify)(ADMDialogRef inDialog, const char*  
inNotifierType);
```

Description

The **SendNotify ()** function sends a notification of type **inNotifierType** to the **inDialog**.

Parameters

inDialog	An ADM dialog.
inNotifierType	The type of notification to send. (See ADMNotifier.h for the notification constants.)

Returns

None.

sADMDialog->SetBoundsRect()

Set the absolute position and size of a dialog

```
void ASAPI (*SetBoundsRect)(ADMDialogRef inDialog, const  
ASRect* inBoundsRect);
```

Description

The **SetBoundsRect ()** function sets the size and position of the dialog window relative to the current screen bounds.

Because the size of the dialog is expressed in screen coordinates, its dimensions must be computed:

```
width = boundsRect.right - boundsRect.left;
height = boundsRect.bottom - boundsRect.top;
```

To move the dialog on the screen, you first get the dialog's bounds rectangle and change it by the move amount.

```
sDialog->GetBoundsRect(myDialog, &boundsRect);
boundsRect.right += relativeMove.h;
boundsRect.left += relativeMove.h;
boundsRect.top += relativeMove.v;
boundsRect.bottom += relativeMove.v;
sDialog->SetBoundsRect(myDialog, &boundsRect);
```

It is simpler to do this using the [sADMDialog->Move\(\)](#) function.

If the dialog is moved off-screen using this function, it will not be forced back onto the screen.

Parameters

inDialog	An ADM dialog.
inBoundsRect	The size and position to make inDialog in screen coordinates. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDialog->GetBoundsRect\(\)](#)
[sADMDialog->Move\(\)](#)
[sADMDialog->GetLocalRect\(\)](#)
[sADMDialog->SetLocalRect\(\)](#)
[sADMDialog->Size\(\)](#)

sADMDialog->SetCancelItemID()

Set the item to cancel a dialog

```
void ASAPI (*SetCancelItemID)(ADMDialogRef inDialog, ASInt32
inItemID);
```

Description

The **SetCancelItemID()** function specifies which item in **inDialog** is the **Cancel** item. The **Cancel** item is notified when the **Esc** key is pressed. On the Macintosh platform, it is also notified when **Command .** is pressed.

Parameters

inDialog	An ADM dialog.
inItemID	ID of the Cancel item in inDialog .

Returns

None.

See also

[sADMDialog->GetCancelItemID\(\)](#)

sADMDialog->SetCursorID()

Set the dialog's cursor ID

```
ASBoolean ASAPI (*SetCursorID)(ADMDialogRef inDialog,  
SPPluginRef inPluginRef, ASInt32 inCursorID);
```

Description

The **SetCursorID()** function sets the cursor to be displayed when the mouse position is inside **inDialog**. The list of built-in ADM cursor IDs is documented in **ADMResource.h**.

NOTE: All negatively numbered cursor IDs are reserved for ADM use only.

NOTE: The object that sets the cursor ID must change it back to be the arrow cursor before some other object can set the cursor ID. When an object sets the cursor ID, it owns it until it resets it to the arrow cursor.

Parameters

inDialog	An ADM dialog.
inPluginRef	Plug-in reference.
inCursorID	ID of the cursor resource. (See ADMResource.h for a list of built-in ADM cursor IDs.)

Returns

None.

See also

[sADMDialog->GetCursorID\(\)](#)

sADMDialog->SetDefaultItemID()

Set the default item of a dialog

```
void WINAPI (*SetDefaultItemID)(ADMDialogRef inDialog, ASInt32
inItemID);
```

Description

The **SetDefaultItemID()** function specifies which item in **inDialog** is the default item. The default item is notified when the **Enter** key is pressed.

Parameters

inDialog	An ADM dialog.
inItemID	ID of the item in inDialog that is the default item.

Returns

None.

See also

[sADMDialog->GetDefaultItemID\(\)](#)

sADMDialog->SetDestroyProc()

Set the ADM destroy function to use for the dialog

```
void WINAPI (*SetDestroyProc)(ADMDialogRef inDialog,
ADMDialogDestroyProc inDestroyProc);
```

Description

The **SetDestroyProc()** function assigns Destroy proc **inDestroyProc** to **inDialog**. **inDestroyProc** will be called when the [sADMDialog->Destroy\(\)](#) function is called for **inDialog**. Use it to free memory and other resources you allocated in your dialog initialization function.

ADM will destroy the dialog and its items, so you do not need to call a default destroy function.

Parameters

inDialog	An ADM dialog.
inDestroyProc	A callback with the following signature (see <code>ADMDialog.h</code>): ADMDialogDestroyProc (ADMDialogRef inDialog);

Returns

None.

See also

[sADMDialog->GetDestroyProc\(\)](#)

sADMDialog->SetDialogName()

Set the dialog name

```
void WINAPI (*SetDialogName)(ADMDialogRef inDialog, const char*
inName);
```

Description

The **SetDialogName()** function sets **inDialog**'s name to **inName**. **inName** is the internal identifier for the dialog. It is the only ID that you can guarantee is unique, assuming that you use a unique name.

Parameters

inDialog	An ADM dialog.
inName	Internal identifier for the dialog. Must be unique. The suggested convention is to use a leading prefix of your company's name, as in MyCompanydialogxxxx .

Returns

None.

See also

[sADMDialog->GetDialogName\(\)](#)
[sADMDialogGroup->GetNamedDialog\(\)](#)

sADMDialog->SetDialogStyle()

Set the style of a dialog

```
void WINAPI (*SetDialogStyle)(ADMDialogRef inDialog,
ADMDialogStyle inDialogStyle);
```

Description

The **SetDialogStyle()** function sets the style of **inDialog** as described for the [sADMDialog->Create\(\)](#) and [sADMDialog->Modal\(\)](#) functions.

NOTE: Currently, this routine is not fully implemented.

Parameters

inDialog	An ADM Dialog.
inDialogStyle	The style of an ADM Dialog (see Dialog Basics: Styles) to which to set inDialog .

Returns

None.

See also

[sADMDialog->GetDialogStyle\(\)](#)

sADMDialog->SetDrawProc()

Set the ADM drawing function to use for the dialog

```
void ASAPI (*SetDrawProc)(ADMDialogRef inDialog,
ADMDialogDrawProc inDrawProc);
```

Description

The **SetDrawProc()** function defines a drawing callback for **inDialog** specified using **inDrawProc**. Within **inDrawProc** you can use the ADM Drawer suite functions to perform standard image operations such as drawing lines and pictures.

See the section [Using Event Callbacks](#) in [Chapter 1, “ADM Overview”](#) for more information.

Parameters

inDialog	An ADM dialog.
inDrawProc	A callback with the following signature (see <code>ADMDialog.h</code>): ADMDialogDrawProc (ADMDialogRef inDialog, ADMDrawerRef inDrawer); The inDrawer argument is passed to the ADM Drawer suite functions to indicate where the imaging is to occur.

Returns

None.

See also

[sADMDialog->GetDrawProc\(\)](#)

sADMDialog->SetFont()

Set the font style

```
void ASAPI (*SetFont)(ADMDialogRef inDialog, ADMFont inFont);
```

Description

The **SetFont()** function sets a dialog's text font to the indicated font style.

Parameters

inDialog	An ADM dialog.
inFont	Font to use. Type: ADMFont (see ADMTypes.h)

Returns

None.

See also

[sADMDialog->GetFont\(\)](#)

sADMDialog->SetForcedOnScreen()

Force the dialog on screen

```
void ASAPI (*SetForcedOnScreen)(ADMDialogRef inDialog,  
ASBoolean inForcedOnScreen);
```

Description

The **SetForcedOnScreen()** function forces the dialog on screen.

Parameters

inDialog	An ADM dialog.
inForcedOnScreen	true if inDialog is forced on screen; false otherwise.

Returns

None.

See also

[sADMDialog->IsForcedOnScreen\(\)](#)

sADMDialog->SetHorizontalIncrement()

Set the horizontal increment to use when resizing a window

```
void ASAPI (*SetHorizontalIncrement)(ADMDialogRef inDialog,
ASInt32 inIncrement);
```

Description

The **SetHorizontalIncrement()** function returns the increment to use when resizing **inDialog** horizontally. By default, a dialog's horizontal increment is **1**, making it sizable to any given width. The dialog's width will always be its minimum width plus a multiple of the horizontal increment.

The horizontal increment of a dialog is useful, for instance, when the dialog has items arranged by column, such as icons in a tool palette. The horizontal increment would be the width of a column. As the dialog is sized, there would never be a column of tool icons partially hidden.

Parameters

inDialog	An ADM dialog.
inIncrement	The amount that will be added to or subtracted from inDialog 's width as the user increases or decreases its size. Default: 1 .

Returns

None.

See also

[sADMDialog->GetHorizontalIncrement\(\)](#)
[sADMDialog->GetVerticalIncrement\(\)](#)
[sADMDialog->SetVerticalIncrement\(\)](#)

sADMDialog->SetLocalRect()

Set the size of a dialog

```
void ASAPI (*SetLocalRect)(ADMDialogRef inDialog, const
ASRect* inLocalRect);
```

Description

The **SetLocalRect()** function sets the local size of **inDialog**. Setting the size of the dialog based on the local rectangle means using a (0,0)-based rectangle of the absolute dimensions. Local rect refers to the client (working) area of the dialog; this contrasts with bounds rect, which refers to the entire dialog.

It is simpler to use the [sADMDialog->Size\(\)](#) function to accomplish the same task as this function.

Parameters

inDialog	An ADM dialog.
inLocalRect	Size of the local rect. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDialog->GetLocalRect\(\)](#)
[sADMDialog->Size\(\)](#)
[sADMDialog->SetBoundsRect\(\)](#)
[sADMDialog->GetBoundsRect\(\)](#)

sADMDialog->SetMask()

Set mask for use with tracker function

```
void ASAPI (*SetMask)(ADMDialogRef inDialog, ADMActionMask  
inMask);
```

Description

The **SetMask()** function sets the mask for controlling which events are received by the tracker.

Parameters

inDialog	An ADM dialog.
inMask	The mask. Type: ADMActionMask (see ADMTypes.h)

Returns

None.

See also

[sADMDialog->GetMask\(\)](#)

sADMDialog->SetMaxHeight()

Set the maximum height of a dialog's window

```
void ASAPI (*SetMaxHeight)(ADMDialogRef inDialog, ASInt32  
inHeight);
```

Description

The **SetMaxHeight()** function sets the maximum height in pixels of resizable dialog **inDialog**. When the user is resizing the dialog, ADM will not allow its height to be greater than this value.

Parameters

inDialog	An ADM dialog.
inHeight	Maximum height (in pixels) of resizable dialog inDialog .

Returns

None.

See also

[sADMDialog->GetMaxHeight\(\)](#)
[sADMDialog->GetMinHeight\(\)](#)
[sADMDialog->SetMinHeight\(\)](#)
[sADMDialog->GetMaxWidth\(\)](#)
[sADMDialog->SetMaxWidth\(\)](#)
[sADMDialog->GetMinWidth\(\)](#)
[sADMDialog->SetMinWidth\(\)](#)

sADMDialog->SetMaxWidth()

Set the maximum width of a dialog's window

```
void ASAPI (*SetMaxWidth)(ADMDialogRef inDialog, ASInt32
inWidth);
```

Description

The **SetMaxWidth()** function sets the maximum width in pixels of resizable dialog **inDialog**. When the user is resizing the dialog, ADM will not allow its width to be greater than this value.

Parameters

inDialog	An ADM dialog.
inWidth	Maximum width (in pixels) of resizable dialog inDialog .

Returns

None.

See also

[sADMDialog->GetMaxWidth\(\)](#)
[sADMDialog->GetMinWidth\(\)](#)
[sADMDialog->SetMinWidth\(\)](#)
[sADMDialog->GetMaxHeight\(\)](#)
[sADMDialog->SetMaxHeight\(\)](#)
[sADMDialog->GetMinHeight\(\)](#)
[sADMDialog->SetMinHeight\(\)](#)

sADMDialog->SetMinHeight()

Set the minimum height of a dialog's window

```
void ASAPI (*SetMinHeight)(ADMDialogRef inDialog, ASInt32  
inHeight);
```

Description

The **SetMinHeight()** function sets the minimum height in pixels of resizable dialog **inDialog**. When the user is resizing the dialog, ADM will not allow its height to be less than this value.

Parameters

inDialog	An ADM dialog.
inHeight	Minimum height (in pixels) of resizable dialog inDialog .

Returns

None.

See also

[sADMDialog->GetMinHeight\(\)](#)
[sADMDialog->GetMaxHeight\(\)](#)
[sADMDialog->SetMaxHeight\(\)](#)
[sADMDialog->GetMinWidth\(\)](#)
[sADMDialog->SetMinWidth\(\)](#)
[sADMDialog->GetMaxWidth\(\)](#)
[sADMDialog->SetMaxWidth\(\)](#)

sADMDialog->SetMinWidth()

Get the minimum width of a dialogs window

```
void ASAPI (*SetMinWidth)(DMDialogRef inDialog, ASInt32  
inWidth);
```

Description

The **SetMinWidth()** function sets the minimum width in pixels of resizable dialog **inDialog**. When the user is resizing the dialog, ADM will not allow its width to be less than this value.

Parameters

inDialog	An ADM dialog.
inWidth	Minimum width (in pixels) of resizable dialog inDialog .

Returns

None.

See also

[sADMDialog->GetMinWidth\(\)](#)
[sADMDialog->GetMaxWidth\(\)](#)
[sADMDialog->SetMaxWidth\(\)](#)
[sADMDialog->GetMinHeight\(\)](#)
[sADMDialog->SetMinHeight\(\)](#)
[sADMDialog->GetMaxHeight\(\)](#)
[sADMDialog->SetMaxHeight\(\)](#)

sADMDialog->SetNotifierData()

Set notification data

```
void WINAPI (*SetNotifierData)(ADMDialogRef inDialog,
    ADMUserData inUserData);
```

Description

The **SetNotifierData()** function sets the notification data of **inDialog**, if any. This can be used for custom notification procedures.

Parameters

inDialog	An ADM dialog.
inUserData	Custom notification data. Type: ADMUserData (see ADMTypes.h)

Returns

None.

See also

[sADMDialog->GetNotifierData\(\)](#)

sADMDialog->SetNotifyProc()

Set the ADM notification function to use for the dialog

```
void WINAPI (*SetNotifyProc)(ADMDialogRef inDialog,  
ADMDialogNotifyProc inNotifyProc);
```

Description

The **SetNotifyProc()** function assigns event notification callback **inNotifyProc** to **inDialog**. See [Using Event Callbacks](#) in [Chapter 1, “ADM Overview”](#) for an example.

Parameters

inDialog	An ADM dialog.
inNotifyProc	<p>A callback with the following signature (see <code>ADMDialog.h</code>):</p> <pre>ADMDialogNotifyProc (ADMDialogRef inDialog, ADMNotifierRef inNotifier);</pre> <p>Use inNotifier to use the functions in the ADM Notifier suite. You can use the ADM Notifier suite functions to determine the type of notification received. The inNotifier argument is passed to the ADM Notifier suite functions to indicate the event for which information is being requested.</p>

Returns

None.

See also

[sADMDialog->GetNotifyProc\(\)](#)

sADMDialog->SetText()

Set the dialog title

```
void WINAPI (*SetText)(ADMDialogRef inDialog, const char*  
inText);
```

Description

The **SetText()** function sets **inDialog**'s text to the indicated C string. A dialog's text property is its title, which is displayed in its window title bar.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

inText	Text for inDialog ' s title.
---------------	-------------------------------------

Returns

None.

See also

[sADMDialog->GetText\(\)](#)
[sADMDialog->GetTextLength\(\)](#)

sADMDialog->SetTrackProc()

Set the ADM tracker function to use for the dialog

```
void ASAPI (*SetTrackProc)(ADMDialogRef inDialog,
ADMDialogTrackProc inTrackProc);
```

Description

The **SetTrackProc()** function defines event tracking callback **inTrackProc** for **inDialog**. See [Using Event Callbacks](#) in [Chapter 1, “ADM Overview”](#) for an example.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

inTrackProc	<p>A callback with the following signature (see <code>ADMDialog.h</code>):</p> <pre>ASBoolean ADMDialogTrackProc (ADMDialogRef inDialog, ADMTrackerRef inTracker);</pre> <p>inTracker is passed to the ADM Tracker suite functions to indicate the event for which information is being requested. This callback returns a boolean—if true, the item receives a notify event when the mouse is released; if false, a notify event will not be received. It always means whether or not to send notification <i>except</i> for keystroke events. In those cases it means whether or not the keystroke was handled.</p>
--------------------	---

Returns

None.

See also

[sADMDialog->GetTrackProc\(\)](#)

sADMDialog->SetUpdateEnabled()

Set the update enabled flag for this dialog

```
void WINAPI (*SetUpdateEnabled)(ADMDialogRef inDialog,  
ASBoolean inEnableUpdate);
```

Description

The **SetUpdateEnabled()** function sets the update-enabled flag for **inDialog**. This function can be used to turn screen updating off and on. Use this feature to increase performance or reduce flickering.

Parameters

inDialog	An ADM dialog.
inEnableUpdate	If set to true , update-enabled flag is set for inDialog . If set to false , it is disabled.

Returns

None.

See also[sADMDialog->IsUpdateEnabled\(\)](#)

sADMDialog->SetUserData()

Set the user data pointer for a dialog

```
void WINAPI (*SetUserData)(ADMDialogRef inDialog, ADMUserData  
inUserData);
```

Description

The **SetUserData()** function sets the 4-byte user value stored with **inDialog**. It is automatically set by the [sADMDialog->Create\(\)](#) and [sADMDialog->Modal\(\)](#) functions.

Parameters

inDialog	An ADM dialog.
ADMUserData	A 4-byte value that ADM keeps with the dialog; you can pass a pointer to a block of memory or some other value and retrieve it later using the sADMDialog->GetUserData() function. Type: ADMUserData (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMDialog->GetUserData\(\)](#)
[sADMDialog->Create\(\)](#)
[sADMDialog->Modal\(\)](#)

sADMDialog->SetVerticalIncrement()

Set the vertical increment to use when resizing a window

```
void ASAPI (*SetVerticalIncrement)(ADMDialogRef inDialog,
ASInt32 inIncrement);
```

Description

The **SetVerticalIncrement()** function sets the increment to use when resizing **inDialog** vertically. By default, a dialog's vertical increment is **1**, making it sizable to any given height. The dialog's width will always be its minimum height plus a multiple of the vertical increment.

The vertical increment of a dialog is useful, for instance, when the dialog has items arranged by rows, such as list. The vertical increment is the height of a list entry. As the dialog is sized, full list entries are added or removed from the list display. There is never a list entry partially displayed or white space at the bottom of the list.

Parameters

inDialog	An ADM dialog.
inIncrement	The amount that will be added to or subtracted from the the dialog's height as the user increases or decreases its size.

Returns

None.

See also

[sADMDialog->GetVerticalIncrement\(\)](#)
[sADMDialog->GetHorizontalIncrement\(\)](#)
[sADMDialog->SetHorizontalIncrement\(\)](#)

sADMDialog->SetWindowRef()

Set the window reference

```
void ASAPI (*SetWindowRef)(ADMDialogRef inDialog, ASWindowRef
inWindowRef);
```


Description

The **SetWindowRef()** function sets the platform window reference for **inDialog**. This reference might be used to draw directly into the window.

On Macintosh, this is the window's **GrafPort**:

```
typedef struct GrafPort *ASWindowRef;
```

On Windows, this is the same as a Windows **HWND**:

```
typedef void * ASWindowRef;
```

Parameters

inDialog	An ADM dialog.
inWindowRef	Platform window reference for inDialog . Type: ASWindowRef (see ASTypes.h)

Returns

None.

See also

[sADMDialog->GetWindowRef\(\)](#)

sADMDialog->Show()

Hide or show a dialog

```
void ASAPI (*Show)(ADMDialogRef inDialog, ASBoolean inShow);
```

Description

The **Show()** function hides or shows **inDialog**. If the dialog is a tabbed dialog combined with other dialogs, the group will be affected.

Parameters

inDialog	An ADM dialog.
inShow	If true , makes inDialog visible; if false , hides inDialog .

Returns

None.

sADMDialog->Size()

Set the dimensions of a dialog

```
void ASAPI (*Size)(ADMDialogRef inDialog, ASInt32 inWidth,
ASInt32 inHeight);
```

Description

The **Size()** function sets the size of the local rect (or client area) to the given width and height. It does not set the bounds rect of the dialog (which includes the entire dialog). If the dialog is sized off the screen using this function, it will not be forced back onto the screen.

Parameters

inDialog	An ADM dialog.
inWidth	Width of the client area (in pixels).
inHeight	Height of the client area (in pixels).

Returns

None.

See also

[sADMDialog->GetBoundsRect\(\)](#)
[sADMDialog->SetBoundsRect\(\)](#)
[sADMDialog->Move\(\)](#)
[sADMDialog->GetLocalRect\(\)](#)
[sADMDialog->SetLocalRect\(\)](#)

sADMDialog->UnregisterItemType()

Remove a custom item type

```
ASErr ASAPI (*UnregisterItemType)(SPPluginRef inPluginRef,
ADMItemType inItemType);
```

Description

The **UnregisterItemType()** function makes a custom item type unavailable. See the [Custom Item Types](#) in [Chapter 1, “ADM Overview”](#) for more information.

Parameters

inPluginRef	Plug-in reference.
inItemType	The custom item type. Type: ADMItemType (see ADMTypes.h)

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred.

See also

[sADMDialog->RegisterItemType\(\)](#)

sADMDialog->Update()

Force an update of a dialog's window

```
void ASAPI (*Update)(ADMDialogRef inDialog);
```

Description

This **Update()** function invalidates **inDialog**'s window and immediately updates its contents. The redraw will occur if **inDialog**'s window is both visible and “dirty.”

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

None.

ADM Help Support

ADM has built-in support for ASHelp, a WinHelp-type help system. ASHelp uses WinHelp file definitions in a cross-platform fashion. Every item has a helpID and the system can operate in contextual fashion. For example, selecting **Command ?** in Macintosh or in **Alt + F1** in Windows lets you click an item and see that item's help resource. For plug-ins to support help files, there must be a Plugin Help location in the **PiPL** resource. The following three functions are used with ASHelp.

NOTE: The Help APIs are deprecated in ADM V2.8.

sADMDialog->GetHelpID()

Get the help ID

```
ASHelpID ASAPI (*GetHelpID)(ADMDialogRef inDialog);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **GetHelpID()** function gets the help ID for **inDialog**.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

The help ID. Type: **ASHelpID** (See **ASHelp.h**)

See also

[sADMDialog->SetHelpID\(\)](#)

sADMDialog->Help()

Calls the help routine for a dialog

```
void ASAPI (*Help)(ADMDialogRef inDialog);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **Help()** function calls the **ASHelp** routine for a specific dialog.

Parameters

inDialog	An ADM dialog.
-----------------	----------------

Returns

None.

sADMDialog->SetHelpID()

Set the help ID

```
void ASAPI (*SetHelpID)(ADMDialogRef inDialog, ASHelpID  
inHelpID);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **SetHelpID()** function sets the help ID for **inDialog**. The **inHelpID** is the resource ID for the **ASHelp** resource.

Parameters

inDialog	An ADM dialog.
inHelpID	The resource ID for the ASHelp resource. Type: ASHelpID (See ASHelp.h)

Returns

None.

See also

[sADMDialog->GetHelpID\(\)](#)

8

The ADM Dialog Group Suite

About the ADM Dialog Group Suite

The ADM Dialog Group suite handles the “grouping” or docking of dialogs, as in a docked palette window. It also provides access to the collection of all dialogs that ADM knows about at any given time.

Accessing the Suite

The ADM Dialog Group suite is referred to as:

```
#define kADMDialogGroupSuite    "ADM Dialog Group Suite"
```

with the version constant:

```
#define kADMDialogGroupSuiteVersion2    2
```

NOTE: Determine the suite version number you are using by examining the `ADMDialogGroup.h` header file.

The suite is acquired as follows:

```
ADMDialogGroupSuite *sADMDialogGroup;  
error = SSPBasic->AcquireSuite(kADMDialogGroupSuite,  
    kADMDialogGroupSuiteVersion2, &sADMDialogGroup);  
if (error) goto . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

The ADM Dialog Group Suite’s Position Code and Group Name

The position code parameter that is set and retrieved using the [sADMDialogGroup->SetDialogGroupInfo\(\)](#) and [sADMDialogGroup->GetDialogGroupInfo\(\)](#) functions is used to restore a dialog’s position within a docked/tabbed group. (Tabbed dialogs can be “floating” or they can be “docked” with other dialogs into a docked/tabbed group.) These functions can also be used to set and retrieve the group name. The dialog group is referred to using the name of the dialog that is the first tab in the top dock of the group.

When docking several palettes, it is necessary to determine which palette is first, second, third, etc., plus where the tab is located (1st, 2nd, etc.). All of these settings are determined by the position code. See `ADMDialogGroup.h` for a description of the values that position code can take.

ADM DialogGroup Suite Functions

sADMDialogGroup->CountDialogs()

Count the number of dialogs in a palette

```
ASErr ASAPI (*CountDialogs) (ASInt32* outCount);
```

Description

The **CountDialogs()** function returns the number of dialogs that ADM currently knows about. Since any number of plug-ins can add any number of dialogs, this function provides a means of determining how many currently exist.

Parameters

outCount	The number of dialogs that ADM currently knows about.
-----------------	---

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

sADMDialogGroup->GetDialogGroupInfo()

Get group name and position code

```
ASErr ASAPI (*GetDialogGroupInfo) (ADMDialogRef inDialog,
const char** outGroupName, ASInt32* outPositionCode);
```

Description

The **GetDialogGroupInfo()** function gets the group name and position code for the dialog group. The group name is the name of the top left tab in the dialog. The position code describes the horizontal and vertical position of this particular tabbed dialog as it appears in a docked palette.

Parameters

inDialog	An ADM docked/tabbed dialog.
outGroupName	inDialog 's group name. See The ADM Dialog Group Suite's Position Code and Group Name .
outPositionCode	inDialog 's position code. See The ADM Dialog Group Suite's Position Code and Group Name .

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

See also

[sADMDialogGroup->SetDialogGroupInfo\(\)](#)

sADMDialogGroup->GetDialogName()

Get the dialog name

```
AS_ERR ASAPI (*GetDialogName) (ADMDialogRef inDialog, const
char** outName);
```

Description

The **GetDialogName()** function gets a reference to the name of **inDialog**. This is the unique name specified in the [sADMDialog->Create\(\)](#) or [sADMDialog->Modal\(\)](#) functions, or that was set using the [sADMDialog->SetDialogName\(\)](#) function.

Parameters

inDialog	An ADM dialog.
outName	Reference to the unique character ID name of inDialog .

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

See also

[sADMDialogGroup->GetNamedDialog\(\)](#)
[sADMDialog->GetDialogName\(\)](#)
[sADMDialog->SetDialogName\(\)](#)
[sADMDialog->Create\(\)](#)
[sADMDialog->Modal\(\)](#)

sADMDialogGroup->GetNamedDialog()

Obtain a specified dialog by name

```
AS_ERR ASAPI (*GetNamedDialog) (const char* inName,
ADMDialogRef* outDialog);
```

Description

The **GetNamedDialog()** function obtains **outDialog** specified by its unique dialog name.

Parameters

inName	Reference to the unique character ID name of inDialog .
outDialog	An ADM dialog.

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

See also

[sADMDialogGroup->GetDialogName\(\)](#)
[sADMDialog->GetDialogName\(\)](#)
[sADMDialog->SetDialogName\(\)](#)
[sADMDialog->Create\(\)](#)
[sADMDialog->Modal\(\)](#)

sADMDialogGroup->GetNthDialog()

Obtain a specific dialog reference

```
ASErr ASAPI(*GetNthDialog) (ASInt32 inIndex, ADMDialogRef*
outDialog);
```

Description

The **GetNthDialog()** function obtains dialog reference **outDialog** for a specific dialog in a docked palette. Use this with the [sADMDialogGroup->CountDialogs\(\)](#) function to iterate through the dialog list.

Parameters

inIndex	0-based index used for iterating through ADM's current list of dialogs.
outDialog	Reference to an ADM dialog in ADM's current list of dialogs.

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

sADMDialogGroup->IsCollapsed()

Check to see if the palette is collapsed

```
ASBoolean ASAPI (*IsCollapsed)(ASInt32 inPositionCode);
```

Description

The **IsCollapsed()** function queries whether a position code indicates that a dialog is collapsed in a docked palette. This function is included in the ADM Dialog Group suite for the sake of completeness—you will probably never use it.

Parameters

inPositionCode	A position code. This is usually obtained using the sADMDialogGroup->GetDialogGroupInfo() function.
-----------------------	--

Returns

true if the dialog is collapsed; **false** otherwise.

See also

[sADMDialogGroup->GetDialogGroupInfo\(\)](#)

Example

```
sADMDialogGroup->GetDialogGroupInfo(myDialog, &buffer, &positionCode );
isCollapsed = sADMDialogGroup->IsCollapsed(positionCode);
```

sADMDialogGroup->IsDockVisible()

Test to make sure that dock is visible

```
ASBoolean ASAPI (*IsDockVisible)(ASInt32 inPositionCode);
```

Description

The **IsDockVisible()** function queries whether a position code indicates that a dialog is currently visible in a docked palette.

Parameters

inPositionCode	A position code. This is usually obtained using the sADMDialogGroup->GetDialogGroupInfo() function.
-----------------------	--

Returns

true if dialog is currently visible in a docked palette; **false** otherwise.

Example

```
sADMDialogGroup->GetDialogGroupInfo( myDialog, &buffer, &positionCode);
isVisible = sADMDialogGroup->IsDockVisible(positionCode);
```

sADMDialogGroup->IsFrontTab()

Check to see if the tab is in front

```
ASBoolean ASAPI (*IsFrontTab)(ASInt32 inPositionCode);
```

Description

The **IsFrontTab()** function queries whether a position code indicates that a dialog is the front tab in a docked palette.

Parameters

inPositionCode	A position code. This is usually obtained using the sADMDialogGroup->GetDialogGroupInfo() function.
-----------------------	--

Returns

true if the dialog is the front tab in a docked palette; **false** otherwise.

Example

```
sADMDialogGroup->GetDialogGroupInfo(myDialog, &buffer, &positionCode);
isFrontTab = sADMDialogGroup->IsFrontTab(positionCode);
```

sADMDialogGroup->IsStandAlonePalette()

Find out whether palette is docked or not

```
ASBoolean ASAPI (*IsStandAlonePalette)(ASInt32
inPositionCode);
```

Description

The **IsStandAlonePalette()** function queries whether or not a position code indicates that a dialog is currently docked.

Parameters

inPositionCode	A position code. This is usually obtained using the sADMDialogGroup->GetDialogGroupInfo() function.
-----------------------	--

Returns

true if the dialog is currently docked; **false** otherwise.

Example

```
sADMDialogGroup->GetDialogGroupInfo(myDialog, &buffer, &positionCode);
isStandalone = sADMDialogGroup->IsStandalonePalette(positionCode);
```

sADMDialogGroup->SetDialogGroupInfo()

Set group name and position code

```
ASErr ASAPI (*SetDialogGroupInfo) (ADMDialogRef inDialog,
const char* inGroupName, ASInt32 inPositionCode);
```

Description

The **SetDialogGroupInfo()** function sets the group name and position code for the dialog group. ADM places the dialog along with any others in the specified group. The position used to display each dialog in a group is determined by its position code.

Parameters

inDialog	An ADM docked/tabbed dialog.
inGroupName	inDialog 's group name. See The ADM Dialog Group Suite's Position Code and Group Name .
inPositionCode	inDialog 's position code. See The ADM Dialog Group Suite's Position Code and Group Name .

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

See also

[sADMDialogGroup->GetDialogGroupInfo\(\)](#)

sADMDialogGroup->SetTabGroup()

Set the tab group to be joined

```
ASErr ASAPI (*SetTabGroup) (ADMDialogRef inDialog, const char*
inTabGroupName, ASBoolean inBringToFront);
```

Description

The **SetTabGroup()** function sets the tab group for **inDialog** to join. **inDialog** is added as the “last” (rightmost) dialog in the group.

Parameters

inDialog	An ADM dialog.
inTabGroupName	The tab group for inDialog to join.
inBringToFront	If true , inDialog is brought to front; if false , inDialog is not brought to front.

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred. See [Appendix D](#) for a list of ADM error codes.

sADMDialogGroup->ShowAllFloatingDialogs()

Show or hide all floating dialogs

```
void ASAPI (*ShowAllFloatingDialogs)(ASBoolean inShow);
```

Description

The **ShowAllFloatingDialogs()** function shows or hides all floating dialogs.

Parameters

inShow	if true , all dialogs are shown; if false , all dialogs are hidden.
---------------	---

Returns

None.

sADMDialogGroup->ToggleAllButNoCloseFloatingDialogs()

Toggle all dialogs except those without close boxes

```
void ASAPI (*ToggleAllButNoCloseFloatingDialogs)();
```

Description

The **ToggleAllButNoCloseFloatingDialogs()** function toggles the visibility of all floating dialogs except those without close boxes.

NOTE: In earlier versions of ADM, **ToggleAllFloatingDialogs()** determined whether or not to also toggle the visibility of all floating dialogs without close boxes (i.e., tool palettes) by checking whether the **Shift** key was pressed. Now this functionality is replaced by separating global toggling of dialogs into two different functions—**ToggleAllButNoCloseFloatingDialogs()** and **ToggleAllFloatingDialogs()**.

Parameters

None.

Returns

None.

See also

[sADMDialogGroup->ToggleAllFloatingDialogs\(\)](#)

sADMDialogGroup->ToggleAllFloatingDialogs()

Toggle the dialogs

```
void ASAPI (*ToggleAllFloatingDialogs)();
```

Description

The **ToggleAllFloatingDialogs()** function toggles the visibility of all floating dialogs.

This function is used to toggle between showing and hiding all of the floating dialogs. The first call shows all dialogs, the second call hides them, the third call shows them all again, and so on. If the floating dialogs are in a mixed state (some hidden), the first will show all of them. If all of them are currently visible, this routine will hide all of them.

NOTE: In earlier versions of ADM, **ToggleAllFloatingDialogs()** determined whether or not to also toggle the visibility of all floating dialogs without close boxes (i.e., tool palettes) by checking whether the **Shift** key was pressed. Now this functionality is replaced by separating global toggling of dialogs into two functions—**ToggleAllButNoCloseFloatingDialogs()** and **ToggleAllFloatingDialogs()**.

Parameters

None.

Returns

None.

See also

[sADMDialogGroup->ToggleAllButNoCloseFloatingDialogs\(\)](#)

9

The ADM Drawer Suite

About the ADM Drawer Suite

The ADM Drawer suite is a set of cross platform imaging functions for use within custom ADM drawer functions. If you create a new type of dialog item or want to embellish an existing one, use the ADM Drawer suite functions to create its appearance.

The functions of the ADM Drawer suite are similar to most platform imaging APIs, but are optimized for user interface work. The suite includes basic imaging (e.g., [sADMDrawer->DrawLine\(\)](#), [sADMDrawer->DrawRect\(\)](#), [sADMDrawer->SetClipRect\(\)](#)) and text handling functions. The color system provides an easy way to specify user interface colors—for instance, two ADM color constants are **kADMDisabledColor** and **kADMButtonDownColor**. Similarly, the fonts available to the text functions are limited to those needed to make dialog items, such as **kADMItalicPaletteFont**. In addition, functions are provided that simplify the implementation of many standard dialog items, such as [sADMDrawer->DrawIcon\(\)](#) and [sADMDrawer->DrawUpArrow\(\)](#). If the drawing capabilities of the ADM Drawer suite are insufficient, you can access a drawing port for using platform imaging functions (or, for internal Adobe development, Adobe's imaging functions).

Accessing the Suite

The ADM Drawer suite is referred to as:

```
#define kADMDrawerSuite          "ADM Drawer Suite"
```

with the version constant:

```
#define kADMDrawerSuiteVersion2    2
```

NOTE: Determine the suite version number you are using by examining the `ADMDrawer.h` header file.

The suite is acquired as follows:

```
ADMDrawerSuite *sADMDrawer;  
error = SSPBasic->AcquireSuite(kADMDrawerSuite,  
                               kADMDrawerSuiteVersion2, &sADMDrawer);  
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

ADM Drawer Functions

ADM drawers are callback functions assigned to ADM objects. Their purpose is to draw on the screen the object to which they are assigned. To specify an ADM drawer function to use with an ADM object, use the assignment functions:

```
sADMDialog->SetDrawProc()
sADMItem->SetDrawProc()
sADMList->SetDrawProc()
sADMHierarchyList->SetDrawProc()
```

For ADM dialogs and ADM items, this assignment function is found in the object suite. Drawer functions for ADM Entry objects are assigned to the list that contains them. All entries in a list have the same drawer function.

All ADM drawer callbacks are similar in definition:

```
typedef void ASAPI (*ADMObjectDrawProc)(ADMObjectRef inObject,
    ADMDrawerRef inDrawer);
```

object is a reference to the dialog, item, entry that is to be drawn. The **ADMDrawerRef** argument is basically a graphics device reference and is used with the functions in this suite to indicate the context for the imaging operation.

All ADM objects have a default drawer function that provides their normal appearance. If your customization to an object is an embellishment of this standard appearance, you can call the default drawer and then modify it (the default drawer for ADM User items does nothing). To call the default drawer, you use a function of the object suite:

```
sADMDialog->DefaultDraw()
sADMItem->DefaultDraw()
sADMListEntry->DefaultDraw()
sADMListEntry->DefaultDraw()
```

You pass the **DefaultDraw()** function the arguments that were passed to your customer drawer function, for instance:

```
void mySquareDrawHandler(ADMItemRef item, ADMDrawerRef drawer) {
    sADMItem->DefaultDraw(item, drawer);
}
```

Using ADM Drawer Functions

The functions in the ADM Drawer suite require an **ADMDrawerRef**, which is the target for the drawing operation. One of the arguments passed to your drawer function is a drawer reference, and it is simply passed to each of the ADM drawer functions:

```
void mySquareDrawHandler(ADMItemRef item, ADMDrawerRef drawer) {
    ASRect boundsRect;
    sADMItem->DefaultDraw(item, drawer);

    sADMDrawer->GetBoundsRect(drawer, &boundsRect);
    boundsRect.top -= 2;
    boundsRect.bottom += 2;
```

```

boundsRect.left -= 2;
boundsRect.right += 2;

sADMDrawer->SetADMColor(drawer, kADMShadowColor)
sADMDrawer->DrawRect(drawer, &boundsRect)
}

```

Fonts and Colors

The ADM Drawer suite has a streamlined model for colors and fonts that offers benefits such as facilitating the design of user interface components with the Adobe “look” and simplifying conformance to platform standards. Both of these qualities help the user have a consistent experience with Adobe applications and their plug-ins.

The normal colors you use in implementing a dialog item have been abstracted. Except for black and white, the keywords for the colors are indicative of the role they play in the user interface. Some of the constants may represent the same color. In addition to helping provide a consistent user experience, another benefit of using the ADM color constants is that your interface adapts to the changing platform interface standards as ADM does. The standard colors used by ADM are listed in **theADMColor struct** in `ADMTypes.h`.

Fonts are simplified from the potential hundreds available to just a few, abstracted so that the correct font is used on a given platform. The standard fonts used by ADM are listed in **ADMFont struct** in `ADMTypes.h`.

The constant names for the fonts are indicative of their purpose. The **kADMDialogFont** font is generally bigger than the **kADMPaletteFont** font and is used for modal dialogs. Floating windows should use the smaller font to reduce the screen area needed by a window that is always present. Font attributes such as size are handled automatically by ADM. ADM uses any required system settings when setting these attributes.

The Drawer Coordinate Space

All drawing is done in coordinates local to the object being drawn, specified relative to the drawer origin. By default, the origin of the drawer is the top-left corner of its object's bounding rectangle. You can redefine the origin if you want, and doing so may simplify some drawing operations.

Pixels are drawn in a Macintosh-like fashion, where a pixel is drawn down and to the right of the indicated coordinate.

Drawing Modes

ADM provides two drawing modes—normal mode and XOR mode—that affect how drawing operations occur (see `ADMDrawer.h`):

```
typedef enum
{
    kADMNormalMode = 0,
    kADMXORMode,
    kADMDummyMode = 0xFFFFFFFF
} ADMDrawMode;
```

In normal mode, a graphics operation overwrites the background entirely. In XOR mode, the background color is inverted when the color of the pixel being drawn is black.



A line drawn in
`kADMNormalMode`
mode



A line drawn in
`kADMXORMode`
mode

FIGURE 9.1 *ADM Drawing Modes*

Graphics commands are also affected by a clipping path. A clipping path is a defined region outside of which graphic operations have no effect.



Horizontal lines



Horizontal lines with
a clipping path

FIGURE 9.2 *Drawing With And Without A Clipping Path*

Clipping paths can be set in several ways: as a rectangle, as a polygon, and by combining multiple rectangles and polygons.

ADM Drawer Suite Functions

sADMDrawer->Clear()

Clear the local rect of the drawer

```
void ASAPI (*Clear)(ADMDrawerRef inDrawer);
```

Description

The **clear()** function fills the area of the ADM object being drawn with the background color.

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

Returns

None.

sADMDrawer->ClearRect()

Clear a rect to the background color

```
void ASAPI (*ClearRect)(ADMDrawerRef inDrawer, const ASRect*  
inRect);
```

Description

The **ClearRect()** function fills the area of the specified rectangle **inRect** with the background color. This is equivalent to using [sADMDrawer->SetADMColor\(\)](#) to set the current color to **kADMBackgroundColor** and doing an [sADMDrawer->FillRect\(\)](#).

Parameters

inDrawer	An ADM drawer.
inRect	The rectangle whose area is to be filled with the background color. Type: ASRect (see ASTypes.h)

Returns

None.

sADMDrawer->Create()

Create an ADM drawer reference

```
ADMDrawerRef ASAPI (*Create)(ASPortRef inPortRef, const  
ASRect* inBoundsRect, ADMFont inFont, ASBoolean inForceRoman);
```

Description

The **Create()** function creates an ADM drawer reference on a port. This supports arbitrary drawing into a port where you want to use the ADM Drawer suite to do the imaging. (ADM does this automatically before it calls a Drawer proc.) For instance, using an ADM dialog reference get a window reference using [sADMDialog->GetWindowRef\(\)](#); then, with the window reference, get the port reference using [sADMDrawer->GetADMWindowPort\(\)](#) and pass the port ref to **Create()**. Use the returned **ADMDrawerRef** with any function in the ADM Drawer suite. The user is responsible for calling [sADMDrawer->Destroy\(\)](#) on any drawers generated by **Create()**.

Parameters

inPortRef	The port reference. Obtained using sADMDrawer->GetADMWindowPort() . Type: ASPortRef (see ASTypes.h)
inBoundsRect	The rect into which to draw. Type: ASRect (see ASTypes.h)
inFont	The font to use. Type: ADMFont (see ADMTypes.h)
inForceRoman	Used to force Roman font on non-roman OS.

Returns

An ADM drawer.

See also

[sADMDrawer->Destroy\(\)](#)
[sADMDialog->GetWindowRef\(\)](#)
[sADMDrawer->GetADMWindowPort\(\)](#)
[sADMDrawer->GetPortRef\(\)](#)
[sADMDrawer->GetUpdateRect\(\)](#)

sADMDrawer->Destroy()

Destroy an ADM drawer

```
void ASAPI (*Destroy)(ADMDrawerRef inDrawer);
```

Description

The **Destroy()** function destroys an **ADMDrawerRef** created with [sADMDrawer->Create\(\)](#).

Parameters

inDrawer	The ADM drawer to be destroyed.
-----------------	---------------------------------

Returns

None.

See also

[sADMDrawer->Create\(\)](#)

sADMDrawer->DrawADMImage() Draw an ADM image

```
void ASAPI (*DrawADMImage)(DMDrawerRef inDrawer, ADMImageRef  
inImage, const ASPoint* inTopLeftPoint);
```

Description

The **DrawADMImage()** function draws the referenced image with the top left of the image anchored to **inTopLeftPoint**. The image is created using the functions in the ADM Image suite.

Parameters

inDrawer	An ADM drawer.
inImage	An image created using functions in the ADM Image suite— sADMImage->Create() , sADMImage->CreateBitmap() , or sADMImage->CreateOffscreen() .
inTopLeftPoint	Anchor point for inImage . Type: ASPoint (see ASTypes.h)

Returns

None.

sADMDrawer->DrawADMImageCentered() Draw a centered ADM image

```
void ASAPI (*DrawADMImageCentered)(ADMDrawerRef inDrawer,  
ADMImageRef inImage, const ASRect* inRect);
```

Description

The **DrawADMImageCentered()** function draws the referenced image with the center of the image anchored to the center of **inRect**. The image is created using the functions in the ADM Image suite.

Parameters

inDrawer	An ADM drawer.
inImage	An image created using functions in the ADM Image suite— sADMImage->Create() , sADMImage->CreateBitmap() , or sADMImage->CreateOffscreen() .
inRect	The rect to whose center inImage is anchored. Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

sADMDrawer->DrawAGMImage()

Draw an AGM Image

```
void ASAPI (*DrawAGMImage)(ADMDrawerRef inDrawer, const struct
_t_ADMAGMImageRecord* inImage, const ASFixedMatrix* inMatrix,
ASInt32 inFlags);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **DrawAGMImage()** function draws an AGM (Adobe Graphics Manager) image. The ADM host application may supply a suite of AGM functions, including AGM image support functions. If not, this function cannot be used.

Parameters

inDrawer	An ADM drawer.
inImage	An image created using AGM.
inMatrix	Fixed matrix into which inImage is drawn. A fixed matrix is a matrix anchored to a specific x/y coordinate. Type: ASFixedMatrix (see <code>ASTypes.h</code>)
inFlags	Currently not implemented. Always 0.

Returns

None.

See also

[sADMDrawer->GetAGMPort\(\)](#)

sADMDrawer->DrawDownArrow()

Draw a triangular down arrow

```
void ASAPI (*DrawDownArrow)(ADMDrawerRef inDrawer, const
ASRect* inRect);
```

Description

The **DrawDownArrow()** function draws a downward-pointing triangular arrow button to fill the area of **inRect**. The arrow drawn is similar to those used in the ADM spinner items (see [Dialog Item Objects](#) in [Chapter 1, “ADM Overview](#)).

The larger of the height or width of the rectangle will determine the arrow's size.

Parameters

inDrawer	An ADM drawer.
inRect	Rect into which to draw the arrow. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawLeftArrow\(\)](#)
[sADMDrawer->DrawRightArrow\(\)](#)
[sADMDrawer->DrawUpArrow\(\)](#)

sADMDrawer->DrawIcon()

Draw an icon at a point

```
void ASAPI (*DrawIcon)(ADMDrawerRef inDrawer, ADMIconRef
inIcon, const ASPoint* inTopLeftPoint);
```

Description

The **DrawIcon()** function places the indicated platform **inIcon** in the ADM window. The top-left corner of the icon is anchored to the **inTopLeftPoint**.

Use the ADM item passed to your draw function to get the plug-in reference; use [sADMItem->GetPluginRef\(\)](#) to do this. Then, to get **inIcon**, use the ADM Icon suite's [sADMIcon->GetFromResource\(\)](#) function.

Parameters

inDrawer	An ADM drawer.
inIcon	The icon to draw. Usually loaded using sADMIcon->GetFromResource() .

inTopLeftPoint	Anchor point for inIcon . Type: ASPoint (See ASTypes.h)
-----------------------	---

Returns

None.

See also

[sADMDrawer->DrawRecoloredIcon\(\)](#)
[sADMDrawer->DrawRecoloredIconCentered\(\)](#)
[sADMDrawer->DrawIconCentered\(\)](#)
[sADMItem->GetPluginRef\(\)](#)
[sADMIcon->GetFromResource\(\)](#)

Example

```

void myOtherStaticPictureHandler(ADMItemRef item, ADMDrawerRef drawer) {
    // this function is also similar to the built in static picture item
    ASPoint boundsRect[2];
    SPPluginRef plugin = sADMItem->GetPluginRef(item);
    ADMIconRef icon = sADMIcon->GetFromResource(plugin,
        kIconID, kIconIndex);

    sADMDrawer->GetBoundsRect(drawer, (ASPoint*)boundsRect);
    sADMDrawer->DrawIcon(drawer, icon, &boundsRect[0]);
}

```

sADMDrawer->DrawIconCentered()

Draw an icon in a rectangle

```

void ASAPI (*DrawIconCentered)(ADMDrawerRef inDrawer,
    ADMIconRef inIcon, const ASRect* inRect);

```

Description

The **DrawIconCentered()** function places the indicated icon in the ADM window. The center of the icon is anchored to the center of **inRect**. Load the icon using the ADM Icon suite.

Parameters

inDrawer	An ADM drawer.
inIcon	The icon to draw. Loaded using sADMIcon->GetFromResource() . See sADMDrawer->DrawIcon() .
inRect	The rect to whose center inIcon is anchored. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawRecoloredIcon\(\)](#)
[sADMDrawer->DrawRecoloredIconCentered\(\)](#)
[sADMDrawer->DrawIcon\(\)](#)
[sADMItem->GetPluginRef\(\)](#)
[sADMIcon->GetFromResource\(\)](#)

sADMDrawer->DrawLeftArrow()

Draw a triangular left arrow

```
void ASAPI (*DrawLeftArrow)(ADMDrawerRef inDrawer, const  
ASRect* inRect);
```

Description

The **DrawLeftArrow()** function draws a left-pointing triangular arrow button to fill the area of **inRect**. The arrow drawn is similar to those used in the ADM spinner items (see [Dialog Item Objects](#) in [Chapter 1, “ADM Overview](#)).

The larger of the height or width of the rectangle will determine the arrow's size.

Parameters

inDrawer	An ADM drawer.
inRect	Rect into which to draw the arrow. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawDownArrow\(\)](#)
[sADMDrawer->DrawRightArrow\(\)](#)
[sADMDrawer->DrawUpArrow\(\)](#)

sADMDrawer->DrawLine()

Draw a line

```
void ASAPI (*DrawLine)(ADMDrawerRef inDrawer, const ASPoint*  
inStartPoint, const ASPoint* inEndPoint);
```

Description

The **DrawLine()** function draws a line 1-point in width in the current color between the **inStartPoint** and **inEndPoint**.

Parameters

inDrawer	An ADM drawer.
inStartPoint	Starting point of line to be drawn. Type: ASPoint (see ASTypes.h)
inEndPoint	Ending point of line to be drawn. Type: ASPoint (see ASTypes.h)

Returns

None.

sADMDrawer->DrawOval()

Draw an unfilled oval

```
void ASAPI (*DrawOval)(ADMDrawerRef inDrawer, const ASRect*
inRect);
```

Description

The **DrawOval()** function draws a line 1-point in width in the current color outlining an oval bounded by **inRect**. The interior of the oval is not filled.

Parameters

inDrawer	An ADM drawer.
inRect	Bounding rectangle for the oval to be drawn. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->FillOval\(\)](#)

sADMDrawer->DrawPolygon()

Draw an unfilled polygon

```
void ASAPI (*DrawPolygon)(ADMDrawerRef inDrawer, const
ASPoint* inPoints, ASInt32 inNumPoints);
```

Description

The **DrawPolygon()** function draws a line 1-point in width in the current color between the **inPoints**. The number of points is specified by the 0-based **inNumPoints** array index. The interior of the polygon is not filled.

Parameters

inDrawer	An ADM drawer.
inPoints	An array of pointers to the points of the polygon to be drawn. Type: ASPoint (see ASTypes.h)
inNumPoints	Zero-based index indicating the number of elements in inPoints .

Returns

None.

See also

[sADMDrawer->SubtractClipPolygon\(\)](#)
[sADMDrawer->FillPolygon\(\)](#)
[sADMDrawer->IntersectClipPolygon\(\)](#)
[sADMDrawer->SetClipPolygon\(\)](#)
[sADMDrawer->UnionClipPolygon\(\)](#)

sADMDrawer->DrawRaisedRect()

Draw a raised rectangle

```
void ASAPI (*DrawRaisedRect)(ADMDrawerRef inDrawer, const  
ASRect* inRect);
```

Description

The **DrawRaisedRect()** function draws a rectangle filled with the background color. The top and left edges of the rectangle are drawn in a lighter gray than the bottom and right edges.



FIGURE 9.3 *A Raised Rectangle*

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

inRect	The size and placement of the raised rectangle. Type: ASRect (see <code>ASTypes.h</code>)
---------------	---

Returns

None.

See also

[sADMDrawer->DrawSunkenRect\(\)](#)

sADMDrawer->DrawRecoloredIcon()

Draw a recolored icon

```
void ASAPI (*DrawRecoloredIcon)(ADMDrawerRef inDrawer,
    ADMIconRef inIcon, const ASPoint* inTopLeftPoint,
    ADMRecolorStyle inStyle);
```

Description

The **DrawRecoloredIcon()** function draws a recolored icon on the screen to indicate, for example, that an icon is disabled. The top left corner of the icon image is anchored to **inTopLeftPoint**. Load the icon using the ADM Icon suite.

Parameters

inDrawer	An ADM drawer.
inIcon	The icon to draw. Usually loaded using sADMIcon->GetFromResource() . See sADMDrawer->DrawIcon() .
inTopLeftPoint	Anchor point for inIcon . Type: ASPoint (See <code>ASTypes.h</code>)
inStyle	Style for the recolored icon. Type: ADMRecolorStyle (see <code>ADMDrawer.h</code>)

Returns

None.

See also

[sADMDrawer->DrawRecoloredIconCentered\(\)](#)
[sADMDrawer->DrawIcon\(\)](#)
[sADMDrawer->DrawIconCentered\(\)](#)
[sADMItem->GetPluginRef\(\)](#)
[sADMIcon->GetFromResource\(\)](#)

sADMDrawer->DrawRecoloredIconCentered()Draw a centered
recolored icon

```
void ASAPI (*DrawRecoloredResPictureCentered)(ADMDrawerRef  
inDrawer, ADMIconRef inIcon, const ASRect* inRect,  
ADMRecolorStyle inStyle);
```

Description

The **DrawRecoloredResPictureCentered()** function draws a centered recolored icon image. This can be useful to, for example, indicate that an icon is disabled. The center of the icon is anchored to the center of **inRect**. Load the icon using the ADM Icon suite.

Parameters

inDrawer	An ADM drawer.
inIcon	The icon to draw. Usually loaded using sADMIcon->GetFromResource() . See sADMDrawer->DrawIcon() .
inRect	The rect to whose center inIcon is anchored. Type: ASRect (see ASTypes.h)
inStyle	Style for the recolored icon. Type: ADMRecolorStyle (see ADMDrawer.h)

Returns

None.

See also

[sADMDrawer->DrawRecoloredIcon\(\)](#)
[sADMDrawer->DrawIcon\(\)](#)
[sADMDrawer->DrawIconCentered\(\)](#)
[sADMItem->GetPluginRef\(\)](#)
[sADMIcon->GetFromResource\(\)](#)

sADMDrawer->DrawRecoloredResPicture()

Draw a recolored image

```
void ASAPI (*DrawRecoloredResPicture)(ADMDrawerRef inDrawer,  
SPPluginRef inPluginRef, ASInt32 inRsrcID, const ASPoint*  
inTopLeftPoint, ADMRecolorStyle inStyle);
```

Description

The **DrawRecoloredResPicture()** function draws a recolored resource image to indicate, for example, that a custom item is disabled. The top left corner of the picture

image is anchored to the `topLeftPoint`. The image is drawn from the platform specific resource.

Parameters

inDrawer	An ADM drawer.
inPluginRef	Plug-in reference. Usually obtained using sADMItem->GetPluginRef() or sADMDialog->GetPluginRef() .
inRsrcID	Resource ID. On Macintosh, the resource type is 'PICT' and on Windows it is a bitmap resource.
inTopLeftPoint	Anchor point for recolored resource image. Type: ASPoint (See ASTypes.h)
inStyle	Style for the recolored icon. Type: ADMRecolorStyle (see ADMDrawer.h)

Returns

None.

See also

[sADMDrawer->DrawRecoloredIcon\(\)](#)
[sADMDrawer->DrawRecoloredIconCentered\(\)](#)
[sADMDrawer->DrawRecoloredResPictureCentered\(\)](#)
[sADMDrawer->GetResPictureBounds\(\)](#)
[sADMDrawer->DrawResPicture\(\)](#)
[sADMDrawer->DrawResPictureCentered\(\)](#)

sADMDrawer->DrawRecoloredResPictureCentered()

Draw a centered recolored image

```
void ASAPI (*DrawRecoloredResPictureCentered)(ADMDrawerRef
inDrawer, SPPluginRef inPluginRef, ASInt32 inRsrcID, const
ASRect* inRect, ADMRecolorStyle inStyle);
```

Description

The **DrawRecoloredResPictureCentered()** function draws a recolored resource image centered on the screen. This can be useful to, for example, indicate that a custom item is disabled. The center of the picture is anchored to the center of **inRect**.

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

inPluginRef	Plug-in reference. Usually obtained using sADMItem->GetPluginRef() or sADMDialog->GetPluginRef() .
inRsrcID	Resource ID. On Macintosh, the resource type is 'PICT' and on Windows it is a bitmap resource.
inRect	The rect to whose center the recolored resource image is anchored. Type: ASRect (see ASTypes.h)
inStyle	Style for the recolored icon. Type: ADMRecolorStyle (see ADMDrawer.h)

Returns

None.

See also

[sADMDrawer->DrawRecoloredIcon\(\)](#)
[sADMDrawer->DrawRecoloredIconCentered\(\)](#)
[sADMDrawer->DrawResPicture\(\)](#)
[sADMDrawer->DrawResPictureCentered\(\)](#)
[sADMDrawer->DrawRecoloredResPicture\(\)](#)
[sADMDrawer->GetResPictureBounds\(\)](#)

sADMDrawer->DrawRect()

Draw an unfilled rectangle

```
void ASAPI (*DrawRect)(ADMDrawerRef inDrawer, const ASRect*  
inRect);
```

Description

The **DrawRect ()** function draws a line 1-point in width in the current color outlining the specified rectangle **inRect**. The interior of the rectangle is not filled.

Parameters

inDrawer	An ADM drawer.
inRect	The rectangle to be drawn. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->FillRect\(\)](#)

sADMDrawer->DrawResPicture()

Draw a picture at a point

```
void ASAPI (*DrawResPicture)(ADMDrawerRef inDrawer,
SPPluginRef inPluginRef, ASInt32 inRsrcID, const ASPoint*
inTopLeftPoint);
```

Description

The **DrawResPicture()** function places a picture stored in the indicated plug-in's resource data in the ADM window. The top-left corner of the picture is anchored to **inTopLeftPoint**. Use the ADM item passed to the draw function to get the plug-in reference.

Parameters

inDrawer	An ADM drawer.
inPluginRef	Plug-in reference. Usually obtained using sADMItem->GetPluginRef() or sADMDialog->GetPluginRef() .
inRsrcID	Resource ID. On Macintosh, the resource type is 'PICT' and on Windows it is a bitmap resource.
inTopLeftPoint	Anchor point for the picture. Type: ASPoint (See ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawRecoloredIcon\(\)](#)
[sADMDrawer->DrawRecoloredIconCentered\(\)](#)
[sADMDrawer->DrawResPictureCentered\(\)](#)
[sADMDrawer->DrawRecoloredResPicture\(\)](#)
[sADMDrawer->DrawRecoloredResPictureCentered\(\)](#)
[sADMDrawer->GetResPictureBounds\(\)](#)

Example

```
void myStaticPictureHandler(ADMItemRef item, ADMDrawerRef drawer) {
    // this function is similar to the built in static picture item
    ASPoint boundsRect[2];
    SPPluginRef plugin = sADMItem->GetPluginRef(item);

    sADMDrawer->GetBoundsRect(drawer, (ASPoint*)boundsRect);

    sADMDrawer->DrawResPicture(drawer, plugin, kPictID,
    &boundsRect[0]);
}
```

sADMDrawer->DrawResPictureCentered()

Draw a picture in a rectangle

```
void ASAPI (*DrawResPictureCentered)(ADMDrawerRef inDrawer,  
SPPluginRef inPluginRef, ASInt32 inRsrcID, const ASRect*  
inRect);
```

Description

The **DrawResPictureCentered()** function places a picture stored in the indicated plug-in's resource data in the ADM window. The center of the picture is anchored to the center of **inRect**. Use the ADM item passed to the draw function to get the plug-in reference. See the example in [sADMDrawer->DrawResPicture\(\)](#).

Parameters

inDrawer	An ADM drawer.
inPluginRef	Plug-in reference. Usually obtained using sADMItem->GetPluginRef() or sADMDialog->GetPluginRef() .
inRsrcID	Resource ID. On Macintosh, the resource type is 'PICT' and on Windows it is a bitmap resource.
inRect	The rect to whose center the picture is anchored. Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDrawer->DrawRecoloredIcon\(\)](#)
[sADMDrawer->DrawRecoloredIconCentered\(\)](#)
[sADMDrawer->DrawResPicture\(\)](#)
[sADMDrawer->DrawRecoloredResPicture\(\)](#)
[sADMDrawer->DrawRecoloredResPictureCentered\(\)](#)
[sADMDrawer->GetResPictureBounds\(\)](#)

sADMDrawer->DrawRightArrow()

Draw a triangular right arrow

```
void ASAPI (*DrawRightArrow)(ADMDrawerRef inDrawer, const  
ASRect* inRect);
```

Description

The **DrawRightArrow()** function draws a right-pointing triangular arrow button to fill the area of **inRect**. The arrow drawn is similar to those used in the ADM spinner items (see [Dialog Item Objects](#) in [Chapter 1, "ADM Overview"](#)).

The larger of the height or width of the rectangle will determine the arrow's size.

Parameters

inDrawer	An ADM drawer.
inRect	Rect into which to draw the arrow. Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDrawer->DrawLeftArrow\(\)](#)
[sADMDrawer->DrawDownArrow\(\)](#)
[sADMDrawer->DrawUpArrow\(\)](#)

sADMDrawer->DrawSunkenRect()

Draw a sunken rectangle

```
void ASAPI (*DrawSunkenRect)(ADMDrawerRef inDrawer, const
ASRect* inRect);
```

Description

The **DrawSunkenRect ()** function draws a rectangle filled with the background color. The top and left edges of the rectangle are drawn in a darker gray than the bottom and right edges.



FIGURE 9.4 A Sunken Rectangle

Parameters

inDrawer	An ADM drawer.
inRect	The size and placement of the sunken rectangle. Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDrawer->DrawRaisedRect\(\)](#)

sADMDrawer->DrawText()

Draw text at a point

```
void ASAPI (*DrawText)(ADMDrawerRef inDrawer, const char*
inText, const ASPoint* inPoint);
```

Description

The **DrawText ()** function draws the C-style string **inText** on the screen at the position **inPoint**.

The last font specified by [sADMDrawer->SetFont\(\)](#) is used or the default font for a window type is used to draw the text. **inPoint** is specified in **inDrawer**'s coordinate space.

Parameters

inDrawer	An ADM drawer.
inText	C-style string to be drawn.
inPoint	The point at which to start drawing the text. Specified in inDrawer 's coordinate space. Type: ASPoint (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawTextCentered\(\)](#)
[sADMDrawer->DrawTextInaBox\(\)](#)
[sADMDrawer->DrawTextLeft\(\)](#)
[sADMDrawer->DrawTextRight\(\)](#)

sADMDrawer->DrawTextCentered()

Draw center justified text in a rectangle

```
void ASAPI (*DrawTextCentered)(ADMDrawerRef inDrawer, const
char* inText, const ASRect* inRect);
```

Description

The **DrawTextCentered()** function draws the C-style string **inText** on the screen center-justified in the area bounded by **inRect**.

The rectangle is specified in the drawer's coordinate space. If the length of the text is greater than the width of the rectangle, the text will be clipped.

Parameters

inDrawer	An ADM drawer.
inText	C-style string to be drawn. If the length of the text is greater than the width of inRect , the text will be clipped.
inRect	The rect into which inText is drawn, center-justified. Specified in inDrawer 's coordinate space. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawText\(\)](#)
[sADMDrawer->DrawTextInaBox\(\)](#)
[sADMDrawer->DrawTextLeft\(\)](#)
[sADMDrawer->DrawTextRight\(\)](#)

sADMDrawer->DrawTextInaBox()

Draw text

```
void ASAPI (*DrawTextInaBox)(ADMDrawerRef inDrawer, const
ASRect* inRect, const char* inText);
```

Description

The **DrawTextInaBox()** function inserts C-style string **inText** into **inRect**. Text flows and is clipped within the bounds of the rectangle of the box.

Parameters

inDrawer	An ADM drawer.
inRect	The box into which to flow inText . Type: ASRect (see ASTypes.h)
inText	The text to flow into inRect .

Returns

None.

See also

[sADMDrawer->DrawText\(\)](#)
[sADMDrawer->DrawTextCentered\(\)](#)
[sADMDrawer->DrawTextLeft\(\)](#)
[sADMDrawer->DrawTextRight\(\)](#)

sADMDrawer->DrawTextLeft()

Draw left justified text in a rectangle

```
void ASAPI (*DrawTextLeft)(ADMDrawerRef inDrawer, const char*  
inText, const ASRect* inRect);
```

Description

The **DrawTextLeft()** function draws the C-style string **inText** on the screen left-justified in the area bounded by **inRect**.

The rectangle is specified in the drawer's coordinate space. If the length of the text is greater than the width of the rectangle, the text will be clipped.

Parameters

inDrawer	An ADM drawer.
inText	C-style string to be drawn. If the length of the text is greater than the width of inRect , the text will be clipped.
inRect	The rect into which inText is drawn, left-justified. Specified in inDrawer 's coordinate space. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawText\(\)](#)
[sADMDrawer->DrawTextCentered\(\)](#)
[sADMDrawer->DrawTextInaBox\(\)](#)
[sADMDrawer->DrawTextRight\(\)](#)

sADMDrawer->DrawTextRight()

Draw right justified text in a rectangle

```
void ASAPI (*DrawTextRight)(ADMDrawerRef inDrawer, const char*  
inText, const ASRect* inRect);
```

Description

The **DrawTextRight()** function draws the C-style string **inText** on the screen right-justified in the area bounded by **inRect**.

The rectangle is specified in the drawer's coordinate space. If the length of the text is greater than the width of the rectangle, the text will be clipped.

Parameters

inDrawer	An ADM drawer.
inText	C-style string to be drawn. If the length of the text is greater than the width of inRect , the text will be clipped.
inRect	The rect into which inText is drawn, right-justified. Specified in inDrawer 's coordinate space. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawText\(\)](#)
[sADMDrawer->DrawTextCentered\(\)](#)
[sADMDrawer->DrawTextInaBox\(\)](#)
[sADMDrawer->DrawTextLeft\(\)](#)

sADMDrawer->DrawUpArrow()

Draw a triangular up arrow

```
void ASAPI (*DrawUpArrow)(ADMDrawerRef inDrawer, const ASRect*
inRect);
```

Description

The **DrawUpArrow()** function draws a upward-pointing triangular arrow button to fill the area of **inRect**. The arrow drawn is similar to those used in the ADM spinner items (see [Dialog Item Objects](#) in [Chapter 1, "ADM Overview"](#)).

The larger of the height or width of the rectangle will determine the arrow's size.

Parameters

inDrawer	An ADM drawer.
inRect	Rect into which to draw the arrow. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawLeftArrow\(\)](#)
[sADMDrawer->DrawDownArrow\(\)](#)
[sADMDrawer->DrawRightArrow\(\)](#)

sADMDrawer->FillOval()

Draw a filled oval

```
void ASAPI (*FillOval)(ADMDrawerRef inDrawer, const ASRect*  
inRect);
```

Description

The **FillOval()** function draws a filled-in oval in the current color bounded by the specified rectangle **inRect**.

Parameters

inDrawer	An ADM drawer.
inRect	Bounding rectangle for the oval to be filled. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->DrawOval\(\)](#)

sADMDrawer->FillPolygon()

Draw a filled polygon

```
void ASAPI (*FillPolygon)(ADMDrawerRef inDrawer, const  
ASPoint* inPoints, ASInt32 inNumPoints);
```

Description

The **FillPolygon()** function draws a filled polygon specified by a 1-point wide line between the **inNumPoints** given **inPoints**.

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

inPoints	An array of pointers to the points of the polygon to be filled. Type: ASPoint (see <code>ASTypes.h</code>)
inNumPoints	Zero-based index indicating the number of elements in inPoints .

Returns

None.

See also

[sADMDrawer->SubtractClipPolygon\(\)](#)
[sADMDrawer->DrawPolygon\(\)](#)
[sADMDrawer->IntersectClipPolygon\(\)](#)
[sADMDrawer->SetClipPolygon\(\)](#)
[sADMDrawer->UnionClipPolygon\(\)](#)

sADMDrawer->FillRect()

Draw a filled rectangle

```
void ASAPI (*FillRect)(ADMDrawerRef inDrawer, const ASRect*
inRect);
```

Description

The **FillRect()** function draws a filled-in rectangle in the current color for **inRect**.

Parameters

inDrawer	An ADM drawer.
inRect	The rectangle to be drawn. Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDrawer->DrawRect\(\)](#)

sADMDrawer->GetADMColor()

Get the ADM color being used

```
ADMColor ASAPI (*GetADMColor)(ADMDrawerRef inDrawer);
```

Description

The **GetADMColor()** function returns the constant of the ADM color currently being used for drawing.

If the last color specified was an RGB color, **GetADMColor()** returns **0xFF** in the lower byte of the 4-byte **ADMColor** value. The 8-bit red, green, and blue values are stuffed into the upper bytes.

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

Returns

The current color. The format for the color is **0xRRGGBBFF**. Type: **ADMColor** (See **ADMTypes.h**)

See also

[sADMDrawer->SetADMColor\(\)](#)
[sADMDrawer->GetRGBColor\(\)](#)
[sADMDrawer->SetRGBColor\(\)](#)

sADMDrawer->GetADMWindowPort() Get an ADM window's port reference

```
ASPortRef ASAPI (*GetADMWindowPort)(ASWindowRef inWindowRef);
```

Description

The **GetADMWindowPort()** function gets a port reference for **inWindowRef**. For instance, use it to obtain an **ASPortRef** to pass to the [sADMDrawer->Create\(\)](#) function.

NOTE: You must release the port obtained by this call with [sADMDrawer->ReleaseADMWindowPort\(\)](#).

Parameters

inWindowRef	Platform window reference. Can be obtained using sADMDialog->GetWindowRef() . Type: ASWindowRef (see ASTypes.h)
--------------------	---

Returns

An ADM port reference. Type: **ASPortRef** (see **ASTypes.h**)

See also

[sADMDrawer->ReleaseADMWindowPort\(\)](#)
[sADMDrawer->Create\(\)](#)
[sADMDialog->GetWindowRef\(\)](#)
[sADMDrawer->GetPortRef\(\)](#)

sADMDrawer->GetAGMPort()

Get an AGM port reference

```
ASAPI struct _t_AGMPort* (*GetAGMPort)(ADMDrawerRef inDrawer);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **GetAGMPort()** function returns a reference to an Adobe Graphics Manager (AGM) port for the drawer. With it, you can draw directly into the window using AGM imaging functions. The ADM host application may supply a suite of AGM functions. If not, this function cannot be used.

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

Returns

AN AGM port.

See also

[sADMDrawer->DrawAGMImage\(\)](#)

sADMDrawer->GetBoundsRect()

Get the bounds of the drawer's object relative to the origin

```
void ASAPI (*GetBoundsRect)(ADMDrawerRef inDrawer, ASRect* outBoundsRect);
```

Description

The **GetBoundsRect()** function gets the bounds of the drawer's object relative to the origin. The bounds can be used to compute the dimensions of the object.

Parameters

inDrawer	An ADM drawer.
outBoundsRect	Bounds of the drawer's object relative to the origin. Type: ASRect (see ASTypes.h)

Returns

None.

Example

```
//computing the dimensions of the object
ASRect BoundsRect;
sADMDrawer->GetBoundsRect(drawer, &BoundsRect);
long height, width;
height = BoundsRect.bottom - BoundsRect.top;
width = BoundsRect.right - BoundsRect.left;
```

sADMDrawer->GetClipRect()

Get the drawer's clip rect

```
void ASAPI (*GetClipRect)(ADMDrawerRef inDrawer, ASRect*
outClipRect);
```

Description

The **GetClipRect()** function gets clipping rectangle **outClipRect**. Drawing operations outside of this rectangle will not have any affect.

If the [sADMDrawer->SetClipPolygon\(\)](#) function has been used, this function will return the bounding box of the clip polygon.

Parameters

inDrawer	An ADM drawer.
outClipRect	The clipping rectangle. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->SetClipRect\(\)](#)
[sADMDrawer->SetClipPolygon\(\)](#)
[sADMDrawer->SubtractClipRect\(\)](#)
[sADMDrawer->UnionClipRect\(\)](#)

sADMDrawer->GetDrawMode()

Get the current drawing mode

```
ADMDrawMode ASAPI (*GetDrawMode)(ADMDrawerRef inDrawer);
```

Description

The **GetDrawMode()** function returns the current drawing mode of the drawer.

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

Returns

The current drawing mode of the drawer—normal or XOR (see [Drawing Modes](#)).
Type: **ADMDrawMode** (see `ADMDrawer.h`).

See also

[sADMDrawer->SetDrawMode\(\)](#)

sADMDrawer->GetFont()

Get the font being used

```
ADMFont ASAPI (*GetFont)(ADMDrawerRef inDrawer);
```

Description

The **GetFont()** function returns the font currently being used for text operations. The return value is one of the **ADMFont** constants.

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

Returns

The font type. Type: **ADMFont** (see `ADMTypes.h`).

See also

[sADMDrawer->SetFont\(\)](#)

sADMDrawer->GetFontInfo()

Get metrics for the current font

```
void ASAPI (*GetFontInfo)(ADMDrawerRef inDrawer, ADMFontInfo* outFontInfo);
```

Description

The **GetFontInfo()** function gets information about the current font. Font metrics are needed, for example, to determine where a text string is to be drawn.

Parameters

inDrawer	An ADM drawer.
outFontInfo	The font metrics for the current font. Type: ADMFontInfo (see <code>ADMDrawer.h</code>)

Returns

None.

sADMDrawer->GetOrigin()

Get the drawer origin

```
void ASAPI (*GetOrigin)(ADMDrawerRef inDrawer, ASPoint*  
outOrigin);
```

Description

The **GetOrigin()** function gets the origin used for the upper left corner of the drawer area. The point returned is relative to the top-left corner of the bounds of the drawer's object.

Parameters

inDrawer	An ADM drawer.
outOrigin	The upper left corner of the drawer area. The point returned is relative to the top-left corner of the bounds of the drawer's object. Type: ASPoint (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDrawer->SetOrigin\(\)](#)

sADMDrawer->GetPortRef()

Get a platform port reference

```
ASPortRef ASAPI (*GetPortRef)(ADMDrawerRef inDrawer);
```

Description

The **GetPortRef()** function returns the platform port reference for the drawer's window. With it, you can draw directly into the window with platform imaging functions.

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

Returns

The platform port reference for the drawer's window. Type: **ASPortRef** (see `ASTypes.h`)

See also

[sADMDrawer->Create\(\)](#)
[sADMDrawer->GetADMWindowPort\(\)](#)

sADMDrawer->GetResPictureBounds() Get the bounds of a resource picture

```
void WINAPI (*GetResPictureBounds)(ADMDrawerRef inDrawer,
SPPluginRef inPluginRef, ASInt32 inRsrcID, ASRect*
outBoundsRect);
```

Description

The **GetResPictureBounds()** function gets the bounds of the indicated picture resource. You might use this to determine if scaling needs to be done before calling one of the draw picture functions. It could also be used to determine where to display a custom image in a dialog list, for example.

Parameters

inDrawer	An ADM drawer.
inPluginRef	Plug-in reference. Usually obtained using sADMItem->GetPluginRef() or sADMDialog->GetPluginRef() .
inRsrcID	Resource ID. On Macintosh, the resource type is 'PICT' and on Windows it is a bitmap resource.
outBoundsRect	Bounds of inRsrcID . Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDrawer->DrawRecoloredResPicture\(\)](#)
[sADMDrawer->DrawRecoloredResPictureCentered\(\)](#)
[sADMDrawer->DrawResPicture\(\)](#)
[sADMDrawer->DrawResPictureCentered\(\)](#)

sADMDrawer->GetRGBColor() Get the values of the RGB color being used

```
void WINAPI (*GetRGBColor)(ADMDrawerRef inDrawer, ASRGBColor*
outColor);
```

Description

The **GetRGBColor()** function gets the RGB values of the color currently being used for drawing. To obtain detailed information about ADM color constants, use [sADMDrawer->SetADMColor\(\)](#) and then get the RGB values with this function.

Parameters

inDrawer	An ADM drawer.
outColor	The RGB values of the color currently being used for drawing. Type: ASRGBColor (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->SetRGBColor\(\)](#)
[sADMDrawer->GetADMColor\(\)](#)
[sADMDrawer->SetADMColor\(\)](#)

sADMDrawer->GetTextRectHeight()

Get the height of the text rect

```
ASInt16 ASAPI (*GetTextRectHeight)(ADMDrawerRef inDrawer,  
ASInt16 inWidth, const char* inText);
```

Description

The **GetTextRectHeight()** function returns the height of a text rect when used to display **inText** in the current font.

Parameters

inDrawer	An ADM drawer.
inWidth	The width of the text rect.
inText	The text to add to a text rect with width inWidth . C-style string.

Returns

The height of a text rect when displaying string **inText** in the current font (in pixels).

See also

[sADMDrawer->GetTextWidth\(\)](#)

sADMDrawer->GetTextWidth()

Get the screen width of a string

```
ASInt32 ASAPI (*GetTextWidth)(ADMDrawerRef inDrawer, const  
char* inText);
```

Description

The **GetTextWidth()** function returns the width of the C-style string **inText** as it would be drawn on the screen using the current font.

Parameters

inDrawer	An ADM drawer.
inText	The text string to be measured.

Returns

Width of **inText** as it would be drawn on the screen using the current font (in pixels).

See also

[sADMDrawer->GetTextRectHeight\(\)](#)

sADMDrawer->GetThisFontInfo()

Get information for a font

```
void WINAPI (*GetThisFontInfo)(ADMFont inFont, ADMFontInfo*
outFontInfo);
```

Description

The **GetThisFontInfo()** function obtains metrics for the specified font.

Parameters

inFont	The font for which metrics are obtained. Type: ADMFont (see ADMTypes.h)
outFontInfo	The font metrics for the current font. Type: ADMFontInfo (see ADMDrawer.h)

Returns

None.

sADMDrawer->GetUpdateRect()

Return rect that needs to be updated

```
void WINAPI (*GetUpdateRect)(ADMDrawerRef inDrawer, ASRect*
outRect);
```

Description

The **GetUpdateRect()** function gets the rect that needs updating. The update rect is optionally specified when a drawer is created. It is used to define the bounds of the area any drawing will be clipped to (i.e., you cannot draw outside this area).

Parameters

inDrawer	An ADM drawer.
outRect	The rect that needs updating. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->Create\(\)](#)

sADMDrawer->IntersectClipPolygon()

Intersect the current clipping area with a polygon

```
void ASAPI (*IntersectClipPolygon)(ADMDrawerRef inDrawer,  
const ASPoint* inPoints, ASInt32 inNumPoints);
```

Description

The **IntersectClipPolygon()** function creates a new clipping area formed by intersecting the current clipping area and the polygon defined by **inPoints**; the number of points in **inPoints** is specified by the 0-based **inNumPoints** array index.

Parameters

inDrawer	An ADM drawer.
inPoints	An array of pointers to the points of the polygon to be created. Type: ASPoint (see ASTypes.h)
inNumPoints	Zero-based index indicating the number of elements in inPoints .

Returns

None.

See also

[sADMDrawer->SubtractClipPolygon\(\)](#)
[sADMDrawer->DrawPolygon\(\)](#)
[sADMDrawer->FillPolygon\(\)](#)
[sADMDrawer->SetClipPolygon\(\)](#)
[sADMDrawer->UnionClipPolygon\(\)](#)

sADMDrawer->IntersectClipRect()

Intersect the current clipping area with a new rectangle

```
void ASAPI (*IntersectClipRect)(ADMDrawerRef inDrawer, const
ASRect* inClipRect);
```

Description

The **IntersectClipRect()** function creates a new clipping area formed by intersecting the current clipping area and the specified rectangle.

Parameters

inDrawer	An ADM drawer.
inClipRect	Rectangle used to create the new clipping area. Type: ASRect (see ASTypes.h)

Returns

None.

sADMDrawer->InvertRect()

Invert the colors of a rectangle

```
void ASAPI (*InvertRect)(ADMDrawerRef inDrawer, const ASRect*
inRect);
```

Description

The **InvertRect()** function inverts the pixels within the specified rectangle. In the simplest case, black pixels will be replaced with white pixels and white with black. Other colors will be replaced by their RGB inverse.

Parameters

inDrawer	An ADM drawer.
inRect	The rect whose colors are to be inverted. Type: ASRect (see ASTypes.h)

Returns

None.

sADMDrawer->ReleaseADMWindowPort()Release an ADM window's
port reference

```
void ASAPI (*ReleaseADMWindowPort)(ASWindowRef inWindowRef,  
ASPortRef inPort);
```

Description

The **ReleaseADMWindowPort()** function releases the **ASPortRef** obtained from the [sADMDrawer->GetADMWindowPort\(\)](#) function.

Parameters

inWindowRef	Platform window reference. Type: ASWindowRef (see ASTypes.h)
inPort	An ADM port reference. Type: ASPortRef (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->GetADMWindowPort\(\)](#)
[sADMDialog->GetWindowRef\(\)](#)

sADMDrawer->SetADMColor()

Set a user interface color to use

```
void ASAPI (*SetADMColor)(ADMDrawerRef inDrawer, ADMColor  
inColor);
```

Description

The **SetADMColor()** function sets a predefined color to be used for drawing (also referred to in this documentation as the “current” color). The ADM color values are defined in the **ADMColor** type (see [ADMTypes.h](#)); they are named according their function in the user interface. By using the predefined ADM colors, platform and user color preferences are respected in your ADM user interface objects.

Parameters

inDrawer	An ADM drawer.
inColor	Sets the current color. The format for the color is 0xRRGGBBFF . Type: ADMColor (See ADMTypes.h)

Returns

None.

See also

[sADMDrawer->GetADMColor\(\)](#)
[sADMDrawer->SetRGBColor\(\)](#)
[sADMDrawer->SetRGBColor\(\)](#)

sADMDrawer->SetClipPolygon()

Specify a polygon to use for a clipping area

```
void ASAPI (*SetClipPolygon)(ADMDrawerRef inDrawer, const
ASPoint* inPoints, ASInt32 inNumPoints);
```

Description

The **SetClipPolygon()** function sets the clipping polygon defined by polygon **inPoints**. The number of points is specified by the 0-based **inNumPoints** array index. Drawing operations outside of this rectangle will not have any affect.

Parameters

inDrawer	An ADM drawer.
inPoints	An array of pointers to the points of the clipping polygon to be created. Type: ASPoint (see ASTypes.h)
inNumPoints	Zero-based index indicating the number of elements in inPoints .

Returns

[sADMDrawer->SubtractClipPolygon\(\)](#)
[sADMDrawer->DrawPolygon\(\)](#)
[sADMDrawer->FillPolygon\(\)](#)
[sADMDrawer->IntersectClipPolygon\(\)](#)
[sADMDrawer->UnionClipPolygon\(\)](#)

sADMDrawer->SetClipRect()

Set the drawer's clip rect

```
void ASAPI (*SetClipRect)(ADMDrawerRef inDrawer, const ASRect*
inClipRect);
```

Description

The **SetClipRect()** function sets the clipping rectangle. Drawing operations outside of this rectangle will not have any affect.

Parameters

inDrawer	An ADM drawer.
inClipRect	The clipping rectangle. Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDrawer->GetClipRect\(\)](#)
[sADMDrawer->SubtractClipRect\(\)](#)
[sADMDrawer->UnionClipRect\(\)](#)

sADMDrawer->SetDrawMode()

Set the drawing mode

```
void ASAPI (*SetDrawMode)(ADMDrawerRef inDrawer, ADMDrawMode  
inDrawMode);
```

Description

The **SetDrawMode()** function sets the drawing mode to be used by future drawing commands.

Parameters

inDrawer	An ADM drawer.
inDrawMode	Specifies the drawing mode of the drawer—normal or XOR (see Drawing Modes). Type: ADMDrawMode (see <code>ADMDrawer.h</code>).

Returns

None.

See also

[sADMDrawer->GetDrawMode\(\)](#)

sADMDrawer->SetFont()

Set the font to be used

```
void ASAPI (*SetFont)(ADMDrawerRef inDrawer, ADMFont inFont);
```

Description

The **SetFont()** function sets the font to be used for subsequent text operations.

Parameters

inDrawer	An ADM drawer.
inFont	Sets the font for use. Type: ADMFont (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMDrawer->GetFont\(\)](#)

sADMDrawer->SetOrigin()

Set the drawer origin

```
void ASAPI (*SetOrigin)(ADMDrawerRef inDrawer, const ASPoint*
inOrigin);
```

Description

The **SetOrigin()** function sets the origin used for the upper left corner of the drawer area. In effect, it subtracts the origin horizontal and vertical values from the coordinates of the drawing commands executed following this call.

Parameters

inDrawer	An ADM drawer.
inOrigin	Sets the upper left corner of the drawer area. The point is relative to the top-left corner of the bounds of the drawer's object. Type: ASPoint (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDrawer->GetOrigin\(\)](#)

sADMDrawer->SetRGBColor()

Set the RGB color to use

```
void ASAPI (*SetRGBColor)(ADMDrawerRef inDrawer, const
ASRGBColor* inColor);
```

Description

The **SetRGBColor()** function is used to specify an arbitrary RGB color that is used when drawing. After using the function, calls to [sADMDrawer->GetADMColor\(\)](#) return

0xFF in the lower byte of the 4-byte **ADMColor** value returned. The 8-bit red, green, and blue values are stuffed into the upper bytes.

Parameters

inDrawer	An ADM drawer.
inColor	Sets the RGB values of the color used for drawing. Type: ASRGBColor (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->GetRGBColor\(\)](#)
[sADMDrawer->GetADMColor\(\)](#)
[sADMDrawer->SetADMColor\(\)](#)

sADMDrawer->SubtractClipPolygon()

Remove a polygon from the existing clip area

```
void ASAPI (*SubtractClipPolygon)(ADMDrawerRef inDrawer, const
ASPoint* inPoints, ASInt32 inNumPoints);
```

Description

The **SubtractClipPolygon()** function creates a new clipping area formed by removing the specified the polygon. The polygon is defined by **inPoints**. The number of points is specified by the 0-based **inNumPoints** array index.

Parameters

inDrawer	An ADM drawer.
inPoints	An array of pointers to the points of the polygon to be subtracted from the current clip area. Type: ASPoint (see ASTypes.h)
inNumPoints	Zero-based index indicating the number of elements in inPoints .

Returns

None.

See also

[sADMDrawer->DrawPolygon\(\)](#)
[sADMDrawer->FillPolygon\(\)](#)

[sADMDrawer->IntersectClipPolygon\(\)](#)
[sADMDrawer->SetClipPolygon\(\)](#)
[sADMDrawer->UnionClipPolygon\(\)](#)

sADMDrawer->SubtractClipRect()

Remove a clip rectangle from the existing clip area

```
void ASAPI (*SubtractClipRect)(ADMDrawerRef inDrawer, const
ASRect* inClipRect);
```

Description

The **SubtractClipRect()** function creates a new clipping area formed by removing the specified rectangle from the current clipping area.

Parameters

inDrawer	An ADM drawer.
inClipRect	Rectangle to be removed from the current clipping area. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMDrawer->GetClipRect\(\)](#)
[sADMDrawer->SetClipRect\(\)](#)
[sADMDrawer->UnionClipRect\(\)](#)

sADMDrawer->UnionClipPolygon()

Combine the existing clip area with a polygon

```
void ASAPI (*UnionClipPolygon)(ADMDrawerRef inDrawer, const
ASPoint* inPoints, ASInt32 inNumPoints);
```

Description

The **UnionClipPolygon()** function creates a new clipping area formed by taking the union of the current clipping area and the polygon defined by **inPoints**. The number of points in the polygon **inPoints** is specified by the 0-based **inNumPoints** array index.

Parameters

inDrawer	An ADM drawer.
-----------------	----------------

inPoints	An array of pointers to the points of the polygon to be used in the union with the current clip area. Type: ASPoint (see <code>ASTypes.h</code>)
inNumPoints	Zero-based index indicating the number of elements in inPoints .

Returns

None.

See also

[sADMDrawer->SubtractClipPolygon\(\)](#)
[sADMDrawer->DrawPolygon\(\)](#)
[sADMDrawer->FillPolygon\(\)](#)
[sADMDrawer->IntersectClipPolygon\(\)](#)
[sADMDrawer->SetClipPolygon\(\)](#)

sADMDrawer->UnionClipRect()

Combine the existing clip area with a new rectangle

```
void ASAPI (*UnionClipRect)(ADMDrawerRef inDrawer, const
ASRect* inClipRect);
```

Description

The **UnionClipRect()** function creates a new clipping area formed by taking the union of the current clipping area and the specified rectangle.

Parameters

inDrawer	An ADM drawer.
inClipRect	Rectangle used in union with current clipping area. Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMDrawer->GetClipRect\(\)](#)
[sADMDrawer->SetClipRect\(\)](#)
[sADMDrawer->SubtractClipRect\(\)](#)

10

The ADM Entry Suite

About the ADM Entry Suite

The ADM Entry suite allows you to create and access ADM Entry objects which are used in conjunction with the ADM List suite. Most of the functions are those common to all ADM objects, such as text access functions. A few are similar to ADM Item objects, such as setting picture IDs, and the remainder are unique to ADM Entry objects—for instance, checking if an entry is selected. This function reference builds on ideas established in [Chapter 1, “ADM Overview”](#).

Accessing the Suite

The ADM Entry suite is referred to as:

```
#define kADMEnterSuite          "ADM Entry Suite"
```

with the version constant:

```
#define kADMEnterSuiteVersion2    2
```

NOTE: Determine the suite version number you are using by examining the `ADMEnter.h` header file.

The suite is acquired as follows:

```
ADMEnterSuite *sADMEnter;  
error = sSPBasic->AcquireSuite(kADMEnterSuite, kADMEnterSuiteVersion2,  
    &sADMEnter);  
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

Initializing An Entry

When you assign a menu ID to a list, ADM creates and initializes the list's ADM Entry objects. They are given an ID and assigned a text string. If you need initialization beyond this, or if you create your own ADM entries, you need to perform these tasks yourself. This can be handled by assigning an **ADMEnterInitProc** (see `ADMList.h`) callback function to the list. It is called for each entry that is created.

What you need to initialize depends on what the entries represent. ADM has a number of standard entry properties which can be set, such as picture IDs. In addition, you can perform your own initialization, such as allocating memory or loading resources. If you do your own initialization, you will likely need to replace the list entries' draw function as well.

NOTE: Unlike ADM dialogs and items, custom handler functions for entries, such as Init and Draw procs, are set for the containing list, not for individual entries. (See [Custom ADM Lists](#) in [Chapter 15, “The ADM List Suite”](#).)

ADM Entry Suite Functions

sADMEEntry->AbortTimer()

Abort timer

```
void ASAPI (*AbortTimer)(ADMEEntryRef inEntry, ADMTimerRef
inTimer);
```

Description

The **AbortTimer()** function aborts a timer procedure. It is used if the event specified by the **inAbortMask** in [sADMEEntry->CreateTimer\(\)](#) occurs or if you destroy your dialog before your timer expires.

Parameters

inEntry	An ADM entry.
inTimer	An ADM timer.

Returns

None.

See also

[sADMEEntry->CreateTimer\(\)](#)

sADMEEntry->Activate()

Make an entry active or inactive

```
void ASAPI (*Activate)(ADMEEntryRef inEntry, ASBoolean
inActivate);
```

Description

The **Activate()** function activates an entry. Activates/deactivates the associated list item.

NOTE: An entry's active and select states are the same.

Parameters

inEntry	An ADM entry.
----------------	---------------

inActivate	If true , the entry is activated; if false , it is inactivated.
-------------------	---

Returns

None.

See also

[sADMEEntry->IsActive\(\)](#)

sADMEEntry->Check()

Set whether an entry is checked

```
void ASAPI (*Check)(ADMEEntryRef inEntry, ASBoolean inCheck);
```

Description

The **Check()** function places a checkmark next to **inEntry**. By default, only menu lists can have checked entries.

NOTE: This state is valid for list box items, though unused. If you are implementing a custom drawer for a list's entries, you could use this value.

Parameters

inEntry	An ADM entry.
inCheck	If true , the entry has a checkmark; if false , it does not have a checkmark.

Returns

None.

See also

[sADMEEntry->IsChecked\(\)](#)

sADMEEntry->Create()

Create a new ADM entry

```
ADMEEntryRef ASAPI (*Create)(ADMListRef inList);
```

Description

The **Create()** function makes a new ADM entry in the indicated **inList**. The ID of the entry is 0 if you do not explicitly set it with the [sADMEEntry->SetID\(\)](#) function. Use [sADMEEntry->Destroy\(\)](#) to dispose of the entry.

Parameters

inList	An ADM list.
---------------	--------------

Returns

The new **ADMEntryRef**.

See also

[sADMEntry->SetID\(\)](#)
[sADMEntry->Destroy\(\)](#)

sADMEntry->CreateTimer()

Create a timer

```
ADMTimerRef ASAPI (*CreateTimer)(ADMEntryRef inEntry, ASUInt32
inMilliseconds, ADMActionMask inAbortMask, ADMEntryTimerProc
inTimerProc, ADMEntryTimerAbortProc inTimerAbortProc, ASInt32
inOptions);
```

The **CreateTimer()** function creates a timer for measuring time between events. Time is kept in milliseconds, with a user-supplied **inTimerProc** and **inTimerAbortProc**. If the delay succeeds (i.e., not aborted) then the **inTimerProc** will be executed. If the action specified by the **inAbortMask** occurs, **inTimerAbortProc** is called.

Parameters

inEntry	An ADM entry.
inMilliseconds	Number of milliseconds in the delay.
inAbortMask	Actions that cause a timer abort. If an action occurs, inTimerAbortProc is called. Maskable actions are listed in <code>ADMTracker.h</code> . Type: ADMActionMask (see <code>ADMTypes.h</code>)
inTimerProc	A callback with the following signature (see <code>ADMEntry.h</code>): <pre>ASBoolean ADMEntryTimerProc(ADMEntryRef inEntry, ADMTimerRef inTimer);</pre> Executed if the delay succeeds. Returns a boolean. If it returns true , then inTimerProc will be called again after inMilliseconds . If it returns false then inTimerProc will no longer be called.
inTimerAbortProc	A callback with the following signature (see <code>ADMEntry.h</code>): <pre>ADMEntryTimerAbortProc(ADMEntryRef inEntry, ADMTimerRef inTimer, ADMAction inAbortAction);</pre> Called if an action specified by inAbortMask occurs. The values for inAbortAction are of type ADMAction and are listed in <code>ADMTracker.h</code> .

inOptions	Currently unused. Always pass 0 (zero).
------------------	---

Returns

The new **ADMTimerRef**.

See also

[sADMEEntry->AbortTimer\(\)](#)

sADMEEntry->DefaultDraw()

Call ADM's default entry draw function

```
void ASAPI (*DefaultDraw)(ADMEEntryRef inEntry, ADMDrawerRef
inDrawer);
```

Description

The **DefaultDraw()** function calls the entry's current default draw function from within your custom entry draw function. The arguments passed to the custom function are passed through to the **DefaultDraw()** call.

Most likely, you will call the default drawing routine within a custom draw function to get the basic appearance of the entry. Your draw function would then add to the entry's appearance. If you completely change the appearance of an entry, you should not call this function.

Set your custom drawer function using [sADMList->SetDrawProc\(\)](#).

Parameters

inEntry	An ADM entry.
inDrawer	An ADM drawer.

Returns

None.

See also

[sADMList->SetDrawProc\(\)](#)

Example

```
void doNothingDrawHandler(ADMEEntryRef inEntry, ADMDrawerRef inDrawer) {
    sADMEEntry->DefaultDraw(entry, drawer);
}
```

sADMEEntry->DefaultNotify()

Call an ADM entry's default notification function

```
void ASAPI (*DefaultNotify)(ADMEEntryRef inEntry,
    ADMNotifierRef inNotifier);
```

Description

The **DefaultNotify()** function calls the default notification function of the entry. Use this within a custom notification callback function. The arguments passed to the custom function are passed through to the **DefaultNotify()** call.

You will always call the default notification function for an entry to get standard behaviors.

Set your custom notifier function using [sADMList->SetNotifyProc\(\)](#).

Parameters

inEntry	An ADM entry.
inNotifier	An ADM notifier.

Returns

None.

See also

[sADMList->SetNotifyProc\(\)](#)

Example

```
void doNothingNotificationHandler(ADMEEntryRef entry, ADMNotifierRef
    notifier) {
    sADMEEntry->DefaultNotify(entry, notifier);
    // Custom behavior would go here...
}
```

sADMEEntry->DefaultTrack()

Call the default tracker function for the ADM entry

```
ASBoolean ASAPI (*DefaultTrack)(ADMEEntryRef inEntry,
    ADMTrackerRef inTracker);
```

Description

The **DefaultTrack()** function calls the default tracking function of the entry. Use it within a custom tracker callback function. The arguments passed to the custom function are passed through to the **DefaultTrack()** call.

The default tracker function handles entry selection, including multiple selections. A custom tracker would be used, for instance, to determine where the mouse-down event occurred so that you could toggle a picture.

You set your custom tracker function using [sADMList->SetTrackProc\(\)](#).

Parameters

inEntry	An ADM entry.
inTracker	An ADM tracker.

Returns

true if tracker successfully called; **false** otherwise.

See also

[sADMList->SetTrackProc\(\)](#)

Example

```
ASBoolean doNothingTrackHandler(ADMLEntryRef entry, ADMTrackerRef
tracker) {
    return sADMLEntry->DefaultTrack(entry, tracker);
}
```

sADMLEntry->Destroy()

Remove an ADM entry from a list

```
void ASAPI (*Destroy)(ADMLEntryRef inEntry);
```

Description

The **Destroy()** function removes an ADM entry from a list. If you use the ADM List suite function [sADMList->SetDestroyProc\(\)](#) to give the entry a custom destroy function, your function will be triggered by this call.

ADM automatically destroys all entries in a list when the ADM dialog is destroyed. Use this function if you are creating and disposing of entries dynamically in response to user actions.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

None.

See also

[sADMList->SetDestroyProc\(\)](#)
[sADMLEntry->Create\(\)](#)

sADMEEntry->Enable()

Enable or disable an entry

```
void ASAPI (*Enable)(ADMEEntryRef inEntry, ASBoolean inEnable);
```

Description

The **Enable()** function enables or disables **inEntry**. An enabled entry can be selected by the user. A disabled entry is unusable and by default appears with grayed text and a grayed icon if it has one.

Parameters

inEntry	An ADM entry.
inEnable	If true , the entry is enabled; if false , it is disabled.

Returns

None.

See also
[sADMEEntry->IsEnabled\(\)](#)

sADMEEntry->GetBoundsRect()

Get the absolute position and size of an entry

```
void ASAPI (*GetBoundsRect)(ADMEEntryRef inEntry, ASRect*  
outBoundsRect);
```

Description

The **GetBoundsRect()** function gets the current size and position of **inEntry** in its containing list's coordinate space.

Parameters

inEntry	An ADM entry.
outBoundsRect	Current size and position of inEntry in its containing list's coordinate space. Type: ASRect (see ASTypes.h)

Returns

None.

sADMEEntry->GetCheckGlyph()

Get check glyph

```
ADMStandardCheckGlyphID ASAPI (*GetCheckGlyph)(ADMEEntryRef
inEntry);
```

Description

The **GetCheckGlyph()** function returns the type of glyph set for **inEntry**. Flyout menus can have hyphens or bullets in addition to checkmarks.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

The type of glyph set for **inEntry**. Type: **ADMStandardCheckGlyphID** (see **ADMEEntry.h**).

See also

[sADMEEntry->SetCheckGlyph\(\)](#)

sADMEEntry->GetDisabledPicture()

Get the picture to display when entry is disabled

```
ADMIconRef ASAPI (*GetDisabledPicture)(ADMEEntryRef inEntry);
```

Description

The **GetDisabledPicture()** function returns the picture that is set to display when **inEntry** is disabled.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

An ADM icon.

See also

[sADMEEntry->SetDisabledPicture\(\)](#)
[sADMEEntry->GetSelectedPicture\(\)](#)
[sADMEEntry->GetSelectedPictureID\(\)](#)
[sADMEEntry->GetPicture\(\)](#)
[sADMEEntry->GetPictureID\(\)](#)
[sADMEEntry->GetDisabledPictureID\(\)](#)

sADMEEntry->GetDisabledPictureID()

Get the entry's disabled picture ID

```
void ASAPI (*GetDisabledPictureID)(ADMEEntryRef inEntry,
ASInt32* outPictureResID, const char** outPictureResName);
```

Description

The **GetDisabledPictureID()** function gets the resource ID of the picture that is set to be displayed for **inEntry** when it is disabled. If the entry does not have a disabled picture, **outPictureResID** returns 0.

Parameters

inEntry	An ADM entry.
outPictureResID	Resource ID for the picture. If the entry does not have a disabled picture, returns 0.
outPictureResName	Picture resource name.

Returns

None.

See also

[sADMEEntry->SetDisabledPictureID\(\)](#)
[sADMEEntry->GetSelectedPicture\(\)](#)
[sADMEEntry->GetSelectedPictureID\(\)](#)
[sADMEEntry->GetPicture\(\)](#)
[sADMEEntry->GetPictureID\(\)](#)
[sADMEEntry->GetDisabledPicture\(\)](#)

sADMEEntry->GetID()

Get the ID of an entry

```
ASInt32 ASAPI (*GetID)(ADMEEntryRef inEntry);
```

Description

The **GetID()** function returns the ID of **inEntry**. When ADM creates entries using a menu resource, it sets the initial ID of each entry to its index + 1. If you create the entry, it initially has an ID of 0.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

ID of **inEntry**.

See also

[sADMEEntry->SetID\(\)](#)

sADMEEntry->GetIndex()

Get the index of an entry

```
ASInt32 ASAPI (*GetIndex)(ADMEEntryRef inEntry);
```

Description

The **GetIndex()** function gets the index, or position, of an entry in the list. For a menu list, the menu entry that the user selected is obtained using this function (see example).

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

The index of **inEntry** in a list. **inEntry** may be the entry selected by the user from a menu (see example).

Example

```
ASInt32 GetListValue(ADMItem theListItem) {  
    ADMList theList = sADMItem->GetList(theListItem);  
    ADMEEntry theEntry = sADMList->GetActiveEntry(theItem);  
    return sADMEEntry->GetIndex(theEntry);  
}
```

sADMEEntry->GetList()

Get the list of an entry

```
ADMListRef ASAPI (*GetList)(ADMEEntryRef inEntry);
```

Description

The **GetList()** function returns a reference to **inEntry**'s containing ADM List object. Once obtained, the ADM List suite functions can be used to access the list.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

An ADM list.

sADMEEntry->GetLocalRect()

Get the size of an entry

```
void ASAPI (*GetLocalRect)(ADMEEntryRef inEntry, ASRect*
outLocalRect);
```

Description

The **GetLocalRect()** function gets the size of **inEntry** in (0,0)-based dimensions.

Parameters

inEntry	An ADM entry.
outLocalRect	The size of inEntry in (0,0)-based dimensions. The bottom and right members of the ASRect structure are the entry's size. Type: ASRect (see ASTypes.h)

Returns

None.

sADMEEntry->GetPicture()

Get the picture

```
ADMIconRef ASAPI (*GetPicture)(ADMEEntryRef inEntry);
```

Description

The **GetPicture()** function returns the picture used to draw **inEntry**.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

An ADM icon.

See also

[sADMEEntry->SetPicture\(\)](#)
[sADMEEntry->GetPictureID\(\)](#)
[sADMEEntry->GetSelectedPicture\(\)](#)
[sADMEEntry->GetSelectedPictureID\(\)](#)
[sADMEEntry->GetDisabledPicture\(\)](#)
[sADMEEntry->GetDisabledPictureID\(\)](#)

sADMEEntry->GetPictureID()

Get the entry's picture ID

```
void ASAPI (*GetPictureID)(ADMEEntryRef inEntry, ASInt32*  
outPictureResID, const char** outPictureResName);
```

Description

This **GetPictureID()** function gets the resource ID of the picture used to draw an entry. If the item does not use a picture, **outPictureResID** returns 0.

Parameters

inEntry	An ADM entry.
outPictureResID	Resource ID for the picture. If the entry does not have a picture, returns 0.
outPictureResName	Picture resource name.

Returns

None.

See also

[sADMEEntry->SetPictureID\(\)](#)
[sADMEEntry->GetPicture\(\)](#)
[sADMEEntry->GetSelectedPicture\(\)](#)
[sADMEEntry->GetSelectedPictureID\(\)](#)
[sADMEEntry->GetDisabledPicture\(\)](#)
[sADMEEntry->GetDisabledPictureID\(\)](#)

sADMEEntry->GetSelectedPicture()

Get the selected picture

```
ADMIConRef ASAPI (*GetSelectedPicture)(ADMEEntryRef inEntry);
```

Description

The **GetSelectedPicture()** function gets the picture to be displayed when **inEntry** is selected.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

An ADM icon.

See also

[sADMEEntry->SetSelectedPicture\(\)](#)
[sADMEEntry->GetSelectedPictureID\(\)](#)
[sADMEEntry->GetPicture\(\)](#)
[sADMEEntry->GetPictureID\(\)](#)
[sADMEEntry->GetDisabledPicture\(\)](#)
[sADMEEntry->GetDisabledPictureID\(\)](#)

sADMEEntry->GetSelectedPictureID()

Get the entry's selected picture ID

```
void ASAPI (*GetSelectedPictureID)(ADMEEntryRef inEntry,
ASInt32* outPictureResID, const char** outPictureResName);
```

Description

The **GetSelectedPictureID()** function gets the resource ID of the picture used to draw **inEntry** when it is selected. If the entry does not have a picture to display when it is selected, **outPictureResID** returns 0.

Parameters

inEntry	An ADM entry.
outPictureResID	Resource ID for the picture. If the entry does not have a disabled picture, returns 0.
outPictureResName	Picture resource name.

Returns

None.

See also

[sADMEEntry->SetSelectedPictureID\(\)](#)
[sADMEEntry->GetSelectedPicture\(\)](#)
[sADMEEntry->GetPicture\(\)](#)
[sADMEEntry->GetPictureID\(\)](#)
[sADMEEntry->GetDisabledPicture\(\)](#)
[sADMEEntry->GetDisabledPictureID\(\)](#)

sADMEEntry->GetText()

Get the entry's text

```
void ASAPI (*GetText)(ADMEEntryRef inEntry, char* outText,
ASInt32 inMaxLength);
```

Description

The **GetText()** function retrieves an entry's text and places it into the already allocated buffer pointed to by **outText**.

Parameters

inEntry	An ADM entry.
outText	Buffer for inEntry 's text.
inMaxLength	Size of the buffer.

Returns

None.

See also

[sADMEEntry->SetText\(\)](#)

sADMEEntry->GetTextLength()

Get the length of the entry's text

```
ASInt32 ASAPI (*GetTextLength)(ADMEEntryRef inEntry;
```

Description

The **GetTextLength()** function gets the number of characters in **inEntry**'s text.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

Number of characters in **inEntry**'s text.

See also

[sADMEEntry->GetText\(\)](#)
[sADMEEntry->SetText\(\)](#)

sADMEEntry->GetUserData()

Get the user data pointer for an entry

```
ADMUserData ASAPI (*GetUserData)(ADMEEntryRef inEntry);
```

Description

The **GetUserData()** function returns the 4-byte user value stored with **inEntry**. The meaning of the value is defined by **inEntry**'s creator. Commonly it is a pointer to a data structure—for instance, several values which are combined to make up the

entry text. For some entries, it might be a simple 4-byte type, such as a long or a fixed number.

Each ADM entry's user data is independent of the other list entries and its list item's data.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

The 4-byte user value stored with **inEntry**. Type: **ADMUserData** (see **ADMTypes.h**)

See also

[sADMEEntry->SetUserData\(\)](#)

sADMEEntry->Invalidate()

Invalidate the area of an entry

```
void ASAPI (*Invalidate)(ADMEEntryRef inEntry);
```

Description

The **Invalidate()** function invalidates **inEntry**'s bounds within the entry. This causes it to be redrawn next time the screen is updated.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

None.

See also

[sADMEEntry->InvalidateRect\(\)](#)
[sADMEEntry->Update\(\)](#)

sADMEEntry->InvalidateRect()

Invalidate the area of a rectangle

```
void ASAPI (*InvalidateRect)(ADMEEntryRef inEntry, const
ASRect* inInvalRect);
```

Description

The **InvalidateRect()** function invalidates the rectangle's bounds within the dialog's window. This causes it to be redrawn next time the screen is updated.

Parameters

inEntry	An ADM entry.
inInvalidRect	The area to be refreshed. Type: ASRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMEEntry->Invalidate\(\)](#)
[sADMEEntry->Update\(\)](#)

sADMEEntry->IsActive()

Get whether or not an entry is active

```
ASBoolean ASAPI (*IsActive)(ADMEEntryRef inEntry);
```

Description

The **IsActive()** function determines whether **inEntry** is currently active. To change its state, use the [sADMEEntry->Activate\(\)](#) function.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

true if **inEntry** is active; **false** otherwise.

See also

[sADMEEntry->Activate\(\)](#)

sADMEEntry->IsChecked()

Get whether an entry is checked

```
ASBoolean ASAPI (*IsChecked)(ADMEEntryRef inEntry);
```

Description

The **IsChecked()** function determines whether **inEntry** is currently checked. To change its state, use the [sADMEEntry->Check\(\)](#) function.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

true if **inEntry** is checked; **false** otherwise.

See also

[sADMEEntry->Check\(\)](#)

sADMEEntry->IsEnabled()

Get whether or not an entry is enabled

```
ASBoolean ASAPI (*IsEnabled)(ADMEEntryRef inEntry);
```

Description

The **IsEnabled()** function determines if **inEntry** is currently enabled. To change its state, use the [sADMEEntry->Enable\(\)](#) function.

A disabled ADM entry is dimmed and unusable.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

true if **inEntry** is enabled; **false** otherwise.

See also

[sADMEEntry->Enable\(\)](#)

sADMEEntry->IsInBounds()

Find out whether an entry is visible

```
ASBoolean ASAPI (*IsInBounds)(ADMEEntryRef inEntry);
```

Description

The **IsInBounds()** function determines whether **inEntry** is visible within the bounds of the list.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

true if **inEntry** is visible; **false** otherwise.

See also

[sADMEEntry->MakeInBounds\(\)](#)

sADMEEntry->IsSelected()

Get whether a list entry is selected

```
ASBoolean ASAPI (*IsSelected)(ADMEEntryRef inEntry);
```

Description

The **IsSelected()** function determines whether **inEntry** is currently selected. To change its state, use the [sADMEEntry->Select\(\)](#) function.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

true if **inEntry** is selected; **false** otherwise.

See also

[sADMEEntry->Select\(\)](#)
[sADMEEntry->GetIndex\(\)](#)

sADMEEntry->IsSeparator()

Set whether an entry is a separator

```
ASBoolean ASAPI (*IsSeparator)(ADMEEntryRef inEntry);
```

Description

The **IsSeparator()** function determines whether **inEntry** is a separator. To change its state, use the [sADMEEntry->MakeSeparator\(\)](#) function.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

true if **inEntry** is a separator; **false** otherwise.

See also

[sADMEEntry->MakeSeparator\(\)](#)

sADMEEntry->LocalToScreenPoint()

Convert an entry point to screen coordinates

```
void ASAPI (*LocalToScreenPoint)(ADMEEntryRef inEntry, ASPoint* ioPoint);
```

Description

The **LocalToScreenPoint()** function converts a point in **inEntry** to a point in the screen's coordinate space.

Parameters

inEntry	An ADM entry.
ioPoint	A point in inEntry . Type: ASPoint (see <i>ASTypes.h</i>)

Returns

None.

See also

[sADMEEntry->LocalToScreenRect\(\)](#)
[sADMEEntry->ScreenToLocalPoint\(\)](#)

sADMEEntry->LocalToScreenRect()

Convert an entry rectangle to screen coordinates

```
void ASAPI (*LocalToScreenRect)(ADMEEntryRef inEntry, ASRect*
ioRect);
```

Description

The **LocalToScreenRect()** function converts a rectangle in **inEntry**'s coordinates to a rectangle in the screen's coordinate space.

Parameters

inEntry	An ADM entry.
ioRect	A rect in inEntry . Type: ASRect (see <i>ASTypes.h</i>)

Returns

None.

See also

[sADMEEntry->LocalToScreenPoint\(\)](#)
[sADMEEntry->ScreenToLocalRect\(\)](#)

sADMEEntry->MakeInBounds()

Make an entry visible

```
void ASAPI (*MakeInBounds)(ADMEEntryRef inEntry);
```


Description

The **MakeInBounds()** function forces **inEntry** to be visible within the bounds of its list.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

None.

See also

[sADMEEntry->IsInBounds\(\)](#)

sADMEEntry->MakeSeparator()

Set an entry to be a separator

```
void ASAPI (*MakeSeparator)(ADMEEntryRef inEntry, ASBoolean  
inSeparator);
```

Description

The **MakeSeparator()** function makes **inEntry** into a separator. Menu lists can have separators that are used to divide their entries into categories.

This state is valid for list box items, though unused. If you are implementing a custom drawer for a list's entries, you could use this value.

Parameters

inEntry	An ADM entry.
inSeparator	Set to true to make inEntry a separator; set to false to indicate that it is not a separator.

Returns

None.

See also

[sADMEEntry->IsSeparator\(\)](#)

sADMEEntry->ScreenToLocalPoint()

Convert a screen point to entry coordinates

```
void ASAPI (*ScreenToLocalPoint)(ADMEEntryRef inEntry, ASPoint*  
ioPoint);
```

Description

The **ScreenToLocalPoint()** function converts a point in screen coordinates to a point relative to **inEntry**.

Parameters

inEntry	An ADM entry.
ioPoint	A point in inEntry . Type: ASPoint (see <i>ASTypes.h</i>)

Returns

None.

See also

[sADMEEntry->LocalToScreenPoint\(\)](#)
[sADMEEntry->ScreenToLocalRect\(\)](#)

sADMEEntry->ScreenToLocalRect()

Convert a screen rectangle to entry coordinates

```
void ASAPI (*ScreenToLocalRect)(ADMEEntryRef inEntry, ASRect*
ioRect);
```

Description

The **ScreenToLocalRect()** function converts a rectangle in screen coordinates to a rectangle in the coordinate space of **inEntry**.

Parameters

inEntry	An ADM entry.
ioRect	A rect in inEntry . Type: ASRect (see <i>ASTypes.h</i>)

Returns

None.

See also

[sADMEEntry->ScreenToLocalPoint\(\)](#)
[sADMEEntry->LocalToScreenRect\(\)](#)

sADMEEntry->Select()

Set whether a list entry is selected

```
void ASAPI (*Select)(ADMEEntryRef inEntry, ASBoolean inSelect);
```

Description

The **Select()** function selects or deselects an entry. In the case of a single selection list, other entries are deselected automatically.

NOTE: For menu lists, an entry's active and selection state are the same.

Parameters

inEntry	An ADM entry.
inSelect	Set to true to select inEntry ; set to false to deselect it.

Returns

None.

See also

[sADMEEntry->IsSelected\(\)](#)

sADMEEntry->SendNotify()

Send a notification to an entry

```
void ASAPI (*SendNotify)(ADMEEntryRef inEntry, const char*  
inNotifierType);
```

Description

The **SendNotify()** function sends a notification of the type **inNotifierType** to **inEntry**.

This API is part of ADM's notification system. Theoretically, you can send any notifier you like to **inEntry**. If you rely on default behavior, probably nothing will be sent. If you have a custom Notify proc, you can use it as you wish. Most likely, the **inNotifierTypes** that would be sent are **kADMIntermediateChangedNotifier** and (possibly) **kADMUserChangedNotifier**.

Parameters

inEntry	An ADM entry.
inNotifierType	Notifier type.

Returns

None.

sADMEEntry->SetCheckGlyph()

Set check glyph

```
void WINAPI (*SetCheckGlyph)(ADMEEntryRef inEntry,  
ADMStandardCheckGlyphID inCheckGlyph);
```

Description

The **SetCheckGlyph()** function sets the type of glyph for **inEntry**. Flyout menus can now have hyphens or bullets in addition to checkmarks.

Parameters

inEntry	An ADM entry.
inCheckGlyph	The type of glyph to set for inEntry . Type: ADMStandardCheckGlyphID (see ADMEEntry.h).

Returns

None.

See also

[sADMEEntry->GetCheckGlyph\(\)](#)

sADMEEntry->SetDisabledPicture()

Set the picture to display when entry is disabled

```
void WINAPI (*SetDisabledPicture)(ADMEEntryRef inEntry,  
ADMIconRef inPicture);
```

Description

The **SetDisabledPicture()** function sets the picture that is used when **inEntry** is disabled.

Parameters

inEntry	An ADM entry.
inPicture	An ADM icon.

Returns

None.

See also

[sADMEEntry->GetDisabledPicture\(\)](#)
[sADMEEntry->SetDisabledPictureID\(\)](#)

```
sADMEEntry->SetPicture()  
sADMEEntry->SetPictureID()  
sADMEEntry->SetSelectedPicture()  
sADMEEntry->SetSelectedPictureID()
```

sADMEEntry->SetDisabledPictureID()

Set the entry's disabled picture ID

```
void ASAPI (*SetDisabledPictureID)(ADMEEntryRef inEntry,  
ASInt32 inPictureResID, const char* inPictureResName);
```

Description

The **SetDisabledPictureID()** function sets the ID of the picture to be displayed for **inEntry** when it is disabled. **inPictureResID** is the ID of a platform picture or icon resource.

If the entry does not have a disabled picture, ADM will gray the default picture to when the entry is disabled.

Parameters

inEntry	An ADM entry.
inPictureResID	Resource ID for the picture.
inPictureResName	Picture resource name.

Returns

None.

See also

```
sADMEEntry->GetDisabledPictureID()  
sADMEEntry->SetDisabledPicture()  
sADMEEntry->SetPicture()  
sADMEEntry->SetPictureID()  
sADMEEntry->SetSelectedPicture()  
sADMEEntry->SetSelectedPictureID()
```

sADMEEntry->SetID()

Set the ID of an entry

```
void ASAPI (*SetID)(ADMEEntryRef inEntry, ASInt32 inEntryID);
```

Description

The **SetID()** function sets the ID of an entry. If you create the entry, it initially has an ID of 0. You should set it within the entry initialization function.

Parameters

inEntry	An ADM entry.
inEntryID	The ID of inEntry .

Returns

None.

See also

[sADMEEntry->GetID\(\)](#)

sADMEEntry->SetPicture()

Set the picture

```
void ASAPI (*SetPicture)(ADMEEntryRef inEntry, ADMIconRef
inPicture);
```

Description

The **SetPicture()** function sets the picture to be displayed for **inEntry**.

Parameters

inEntry	An ADM entry.
inPicture	An ADM icon.

Returns

None.

See also

[sADMEEntry->GetPicture\(\)](#)
[sADMEEntry->SetPictureID\(\)](#)
[sADMEEntry->SetDisabledPicture\(\)](#)
[sADMEEntry->SetDisabledPictureID\(\)](#)
[sADMEEntry->SetSelectedPicture\(\)](#)
[sADMEEntry->SetSelectedPictureID\(\)](#)

sADMEEntry->SetPictureID()

Set the entry's picture ID

```
void ASAPI (*SetPictureID)(ADMEEntryRef inEntry, ASInt32
inPictureResID, const char* inPictureResName);
```

Description

The **SetPictureID()** sets the ID for the picture to be displayed for **inEntry**. The **inPictureResID** is the ID of a platform picture or icon resource.

For both list box entries and menu list entries, if a picture is set the picture will appear on the left side of the entry. The item text will appear to the right of the picture.

Parameters

inEntry	An ADM entry.
inPictureResID	Resource ID for the picture.
inPictureResName	Picture resource name.

Returns

None.

See also

[sADMEEntry->GetPictureID\(\)](#)
[sADMEEntry->SetPicture\(\)](#)
[sADMEEntry->SetDisabledPicture\(\)](#)
[sADMEEntry->SetDisabledPictureID\(\)](#)
[sADMEEntry->SetSelectedPicture\(\)](#)
[sADMEEntry->SetSelectedPictureID\(\)](#)

sADMEEntry->SetSelectedPicture()

Set the selected picture

```
void ASAPI (*SetSelectedPicture)(ADMEEntryRef inEntry,  
    ADMIconRef inPicture);
```

Description

The **SetSelectedPicture()** function sets the picture to be displayed when **inEntry** is selected. If the entry does not have a picture to use when **inEntry** is selected, ADM inverts the default picture to show that it is selected.

Parameters

inEntry	An ADM entry.
inPicture	An ADM icon.

Returns

None.

See also

[sADMEEntry->GetSelectedPicture\(\)](#)
[sADMEEntry->SetSelectedPictureID\(\)](#)
[sADMEEntry->SetPicture\(\)](#)
[sADMEEntry->SetPictureID\(\)](#)
[sADMEEntry->SetDisabledPicture\(\)](#)
[sADMEEntry->SetDisabledPictureID\(\)](#)

sADMEEntry->SetSelectedPictureID()

Set the entry's selected picture ID

```
void ASAPI (*SetSelectedPictureID)(ADMEEntryRef inEntry,
ASInt32 inPictureResID, const char* inPictureResName);
```

Description

The **SetSelectedPictureID()** function sets the picture ID for the picture to be displayed for **inEntry** when is selected. The **inPictureResID** is the ID of a platform picture or icon resource. If the entry does not have a picture to use when **inEntry** is selected, ADM will invert the default picture to show that it is selected.

Parameters

inEntry	An ADM entry.
inPictureResID	Resource ID for the picture.
inPictureResName	Picture resource name.

Returns

None.

See also

[sADMEEntry->GetSelectedPictureID\(\)](#)
[sADMEEntry->SetSelectedPicture\(\)](#)
[sADMEEntry->SetPicture\(\)](#)
[sADMEEntry->SetPictureID\(\)](#)
[sADMEEntry->SetDisabledPicture\(\)](#)
[sADMEEntry->SetDisabledPictureID\(\)](#)

sADMEEntry->SetText()

Set the entry's text

```
void ASAPI (*SetText)(ADMEEntryRef inEntry, const char*
inText);
```


Description

The **SetText ()** function sets **inEntry**'s text to the indicated C string. An entry's text is used for the platform menu item text or list item text. If an entry has a picture ID, the text is displayed to the right of the picture.

Parameters

inEntry	An ADM entry.
inText	Text for inEntry .

Returns

None.

See also

[sADMEEntry->GetText\(\)](#)

sADMEEntry->SetUserData()

Set the user data pointer for an entry

```
void ASAPI (*SetUserData)(ADMEEntryRef inEntry, ADMUserData  
inUserData);
```

Description

The **SetUserData ()** function sets the 4-byte user value stored with **inEntry**. Each entry has its own user data. If you want to store user data for all entries, use the list's ADM item's user data.

To get the item's user data, get the entry's list and then get the list's item reference. With this you can get the user data directly.

Parameters

inEntry	An ADM entry.
inUserData	The 4-byte user value stored with inEntry . Type: ADMUserData (see ADMTypes.h)

Returns

None.

See also

[sADMEEntry->GetUserData\(\)](#)

sADMEEntry->Update()

Force an update of an entry

```
void ASAPI (*Update)(ADMEEntryRef inEntry);
```

Description

The **Update()** function invalidates the bounds rectangle of **inEntry** and immediately updates its contents. The redraw will occur if **inEntry**'s bounds rect is both visible and "dirty."

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

None.

See also

[sADMEEntry->Invalidate\(\)](#)
[sADMEEntry->InvalidateRect\(\)](#)

ADM Help Support

ADM has built-in support for ASHelp, a WinHelp-type help system. ASHelp uses WinHelp file definitions in a cross-platform fashion. Every item has a helpID and the system can operate in contextual fashion. For example, selecting **Command ?** in Macintosh or in **Alt + F1** in Windows lets you click an item and see that item's help resource. For plug-ins to support help files, there must be a Plugin Help location in the **PiPL** resource. The following three functions are used with ASHelp.

NOTE: These APIs are deprecated in ADM V2.8.

sADMEEntry->GetHelpID()

Get the help ID of an entry

```
ASHelpID ASAPI (*GetHelpID)(ADMEEntryRef inEntry);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **GetHelpID()** function gets the help ID for **inEntry**.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

The help ID. Type: **ASHelpID** (See **ASHelp.h**)

See also

[sADMEEntry->SetHelpID\(\)](#)

sADMEEntry->Help()

Calls the help routine associated with an entry

```
void ASAPI (*Help)(ADMEEntryRef inEntry);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **Help()** function calls the help for **inEntry**.

Parameters

inEntry	An ADM entry.
----------------	---------------

Returns

None.

sADMEEntry->SetHelpID()

Set the help ID

```
void ASAPI (*SetHelpID)(ADMEEntryRef inEntry, ASHelpID  
inHelpID);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **SetHelpID()** function sets the help ID for **inEntry**. **inHelpID** is the resource ID for the ASHelp resource.

Parameters

inEntry	An ADM entry.
inHelpID	The resource ID for the ASHelp resource. Type: ASHelpID (See ASHelp.h)

Returns

None.

See also[sADMEEntry->GetHelpID\(\)](#)

11

The ADM Hierarchy List Suite

About the ADM HierarchyList Suite

The ADM Hierarchy List suite allows you to access ADM Hierarchy List objects and ADM List Entry objects. Since an ADM Hierarchy List object is an extended property of a standard ADM Item object, this suite lacks many of the functions common to ADM objects; however, you can access the a hierarchy list's ADM item and do common operations on it. Using functions in this suite, you can initialize the hierarchy list, and you can create, destroy, customize, and iterate through the ADM list entries of a hierarchy list. The Hierarchy List suite is used in conjunction with the ADM List Entry suite to further access list related information.

NOTE: The relationship between ADM Hierarchy List objects and ADM List Entry objects is the same as that between ADM List objects and ADM Entry objects—that is, list entries are the elements of a hierarchy list. Note that list entries themselves may be hierarchy lists with list entry children of their own.

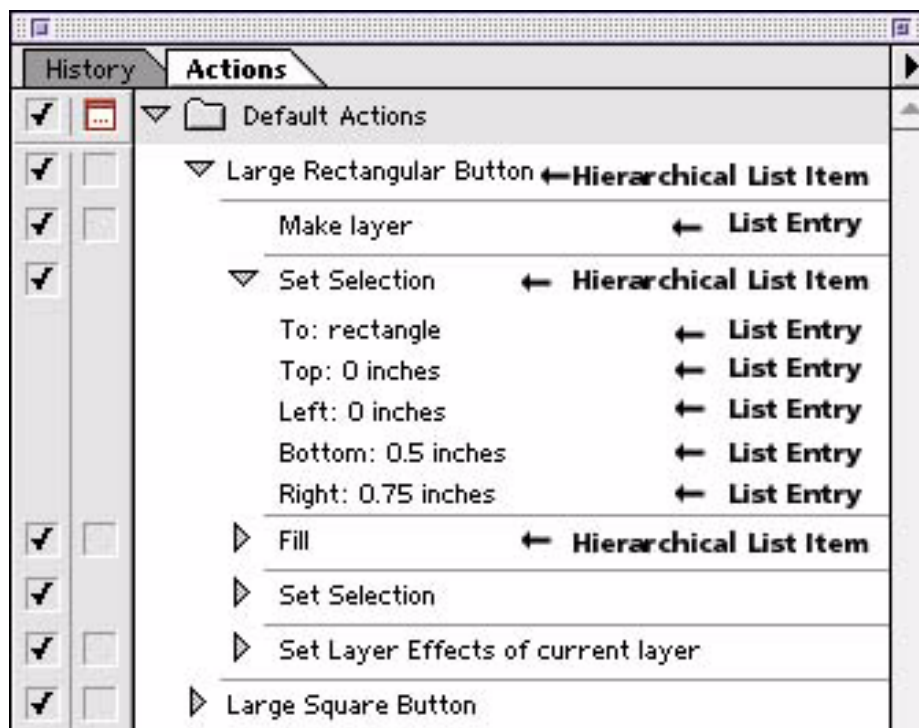


FIGURE 11.1 AN ADM Hierarchy List

Accessing the Suite

The ADM Hierarchy List suite is referred to as:

```
#define kADMHierarchyListSuite          "ADM Hierarchy List Suite"
```

with the version constant:

```
#define kADMHierarchyListSuiteVersion2    2
```

NOTE: Determine the suite version number you are using by examining the `ADMHierarchyList.h` header file.

The suite is acquired as follows:

```
ADMListSuite *sADMHierarchyList;
error = sSPBasic->AcquireSuite(kADMHierarchyListSuite,
    kADMHierarchyListSuiteVersion2, &sADMHierarchyList);
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

ADM Hierarchy Lists and List Entries

ADM Hierarchy List objects are used by ADM Item objects to provide a list of expandable choices, including list boxes, popup lists, popup menus, spin edit popups, and text edit popups. An ADM hierarchy list is composed of ADM list entries.

ADM hierarchy lists do not have many standard properties, such as a plug-in and bounds. Rather, these are defined using the ADM hierarchy list's item. To access them, use [sADMHierarchyList->GetItem\(\)](#) to get the item owning the list and then use the ADM Item suite functions with the returned item reference.

ADM hierarchy lists have special properties, such as a menu resource ID and a group of list entries. ADM list entries have additional properties, including an index and a selected state. These entry properties are used by the ADM Hierarchy List suite to access the entries. The index is the position of the entry in list. The selected state indicates the user has selected the item (others may be selected in the case of a multi-select list).

ADM Hierarchy List Recipes

To get the hierarchy list object for a list entry, you use the [sADMListEntry->GetItem\(\)](#) function:

```
ADMHierarchyListRef theHierarchyList =
    sADMListEntry->GetList(theListEntry);
```

Once this is done you can use the ADM Hierarchy suite functions.

To initialize a list, assign it a menu resource ID:

```
sADMHierarchyList->SetMenuID(theItemsList, gPlugInRef, 16000,
    "Choices");
```

You can also create each entry with the ADM Hierarchy List suite's `sADMHierarchyList->InsertEntry()` function followed by the ADM List Entry suite's `sADMListEntry->SetText()` and `sADMListEntry->SetID()` functions:

```
for (index = 0; index < kNumberEntries; index++) {
    char menuText[255];
    ADMListEntryRef entry = sADMHierarchyList-
>InsertListEntry(theItemList, index);
    sBasic->GetIndexString(thePlugin, 16000, index, menuText, 255);
    sADMListEntry->SetText(entry, menuText);
    sADMListEntry->SetID(entry, index);
}
```

Note that list indices are 0-based.

To get the currently selected item of a single selection list, use the `sADMHierarchyList->GetActiveEntry()` function and then get the entry's index:

```
ASInt32 GetHierarchyListValue(ADMItem theListItem) {
    ADMHierarchyList theList = sADMItem->GetList(theListItem);
    ADMListEntry theEntry = sADMHierarchyList->GetActiveEntry(theList);
    return sADMListEntry->GetID(theEntry);
}
```

To get each selected item in a multiple selection list, get the selection count and iterate through the selections:

```
ASInt32 count = NumberOfSelectedEntries(theList);
for (index = 0; index < count; index++) {
    ADMListEntryRef entry = sADMHierarchyList-
>IndexSelectedEntry(theList, index);
    doSomethingToSelectedEntry(theEntry);
}
```

Custom ADM Hierarchy Lists

You can customize an ADM Hierarchy List object just as you can customize other ADM items. This is done by defining one or more of the event handler functions. Because ADM hierarchy lists are closely linked to ADM list entries, the process is slightly different.

The ADM hierarchy list doesn't actually have its own event handler functions. If you need to do something to the list as a whole in a handler, set the handler function for the list—for instance, to annotate the list, set the drawer function the list. These are assigned using the ADM Item suite.

If you need to change the behavior of the hierarchy list at a lower level, you can set the handler functions of the list's entries; for instance, to change how each list entry draws, you would set the drawer function for the list's entries. This is done at the hierarchy list level, using the ADM Hierarchy List suite, and affects all the list entries in a list. You cannot directly set a handler function for an individual list entry; a custom handler function for a hierarchy list must work for all its list entries.

To use the default behavior for a hierarchy list item, you use the ADM Item suite functions. To use the default behavior for a list entry, you use functions in the ADM List Entry suite; not the ADM Hierarchy List suite.

ADM Hierarchy List Suite Functions

sADMHierarchyList->DeselectAll()

Deselect all items in the hierarchy list

```
void ASAPI (*DeselectAll)(ADMHierarchyListRef inList);
```

Description

The **DeselectAll()** function deselects all entries in **inList**.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

None.

sADMHierarchyList->FindEntry()

Get a hierarchy list entry by text

```
ADMListEntryRef ASAPI (*FindEntry)(ADMHierarchyListRef inList,
const char* inText);
```

Description

The **FindEntry()** function gets a reference to a hierarchy list entry based on its text. ADM searches **inList** for an entry with text that matches **inText** and returns that entry. If no match is found, **NULL** is returned.

NOTE: This function searches not only root level entries, but also children (recursively).

Parameters

inList	An ADM hierarchy list.
inText	The text to search for.

Returns

The ADM list entry that matches **inText**. If no match is found, **NULL** is returned.

sADMHierarchyList->GetActiveEntry()

Get the selected list entry

```
ADMListEntryRef ASAPI (*GetActiveEntry)(ADMHierarchyListRef
inList);
```

Description

The **GetActiveEntry()** function returns a reference to the currently selected list entry in **inList**. If **inList** has multiple selections, it returns the first selection. If a child entry is selected, its parent is effectively selected.

To get all the selected entries of multiple selection list, use the [sADMHierarchyList->IndexSelectedEntry\(\)](#) function.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The active ADM list entry.

See also

[sADMHierarchyList->IndexSelectedEntry\(\)](#)

sADMHierarchyList->GetActiveLeafEntry()

Get the active leaf entry

```
ADMListEntryRef ASAPI
(*GetActiveLeafEntry)(ADMHierarchyListRef inList);
```

Description

The **GetActiveLeafEntry()** function returns a reference to the currently selected leaf entry in **inList**. An entry that has no list attached to it is a leaf. If the entry has a list attached it is referred to as just an entry.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The current selected leaf entry in **inList**.

sADMHierarchyList->GetDestroyProc()

Get the ADM destroy function being used for the hierarchy list's entries

```
ADMListEntryDestroyProc ASAPI
(*GetDestroyProc)(ADMHierarchyListRef inList);
```

Description

The **GetDestroyProc()** gets the destroy function used for the entries of **inList**.

NOTE: Because ADM calls the entry's destroy function when it is removed from its list, you should not call the returned function directly.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The destroy function used for the entries of **inList**. If you have not called [sADMHierarchyList->SetDestroyProc\(\)](#), returns **NULL** (not the default Destroy proc).
Type: **ADMListEntryDestroyProc** (see `ADMHierarchyList.h`)

See also

[sADMHierarchyList->SetDestroyProc\(\)](#)

sADMHierarchyList->GetDivided()

Find out whether a list is a divided list

```
ASBoolean ASAPI (*GetDivided)(ADMHierarchyListRef inList);
```

Description

The **GetDivided()** function finds out whether **inList** is a divided list.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

true if **inList** is a divided list; **false** otherwise.

See also

[sADMHierarchyList->SetDivided\(\)](#)

sADMHierarchyList->GetDrawProc()

Get the ADM drawing function being used for the hierarchy list's entries

```
ADMLEntryDrawProc ASAPI (*GetDrawProc)(ADMHierarchyListRef  
inList);
```

Description

The **GetDrawProc()** function gets the draw function being used for a hierarchy list's entries.

Rather than getting and calling a draw function in this fashion, you are more likely to use the [sADMLEntry->DefaultDraw\(\)](#) function.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The draw function used for the entries of **inList**. If you have not called [sADMHierarchyList->SetDrawProc\(\)](#), returns **NULL** (not the default Drawer proc).

Type: **ADMLEntryDrawProc** (see `ADMHierarchyList.h`)

sADMHierarchyList->GetEntry()

Get a list entry by ID

```
ADMLEntryRef ASAPI (*GetEntry)(ADMHierarchyListRef inList,  
ASInt32 inEntryID);
```

Description

The **GetEntry()** function is used to get a reference to the hierarchy list entry with the indicated ID. If no match is found, **NULL** is returned.

Each ADM list entry object has an ID, which can be obtained with the [sADMLEntry->GetID\(\)](#) function. If you keep this ID, you can pass it to this function at a later time and retrieve the item.

NOTE: This function only works for root level entries—children of root level entries are not searched.

Parameters

inList	An ADM hierarchy list.
inEntryID	ADM hierarchy list entry ID to search for.

Returns

The hierarchy list entry with the indicated ID. If no match is found, **NULL** is returned.

See also[sADMListEntry->GetID\(\)](#)

sADMHierarchyList->GetEntryHeight()

Get the height of a hierarchy list's entry

```
ASInt32 ASAPI (*GetEntryHeight)(ADMHierarchyListRef inList);
```

Description

The **GetEntryHeight()** function returns the height of a hierarchy list's entry. All entries have the same height.

Given this value and the local rectangle size returned by [sADMHierarchyList->GetLocalRect\(\)](#), you can calculate the number of rows that appear in the list.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The height of a hierarchy list's entry.

See also

[sADMHierarchyList->GetEntryWidth\(\)](#)
[sADMHierarchyList->GetLocalRect\(\)](#)
[sADMHierarchyList->SetEntryHeight\(\)](#)
[sADMHierarchyList->SetEntryHeightRecursive\(\)](#)

sADMHierarchyList->GetEntryTextRect()

Get the edit text rectangle for a hierarchy list

```
void ASAPI (*GetEntryTextRect)(ADMHierarchyListRef inList,  
ASRect* outRect);
```

Description

The **GetEntryTextRect()** function returns the location of the edit-in-place text item of **inList**.

Parameters

inList	An ADM hierarchy list.
outRect	Location of the edit-in-place text item of inList . Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMHierarchyList->SetEntryTextRect\(\)](#)
[sADMHierarchyList->SetEntryTextRectRecursive\(\)](#)

sADMHierarchyList->GetEntryWidth()

Get the width of a column of hierarchy list entries

```
ASInt32 ASAPI (*GetEntryWidth)(ADMHierarchyListRef inList);
```

Description

The **GetEntryWidth()** function returns the width of a column if **inList** is part of a listbox item and has the style **kADMTileListBoxStyle**. For any other list style, this function returns the width of the single column of the list.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

Width of a column if **inList** is part of a listbox item and has the style **kADMTileListBoxStyle**; for any other list style, returns the width of the single column of the list.

NOTE: If the entry width has been set (using [sADMHierarchyList->SetEntryWidth\(\)](#)), this function returns that value. Otherwise, this function returns the width of the list itself.

See also

[sADMHierarchyList->GetEntryHeight\(\)](#)
[sADMHierarchyList->SetEntryWidth\(\)](#)
[sADMHierarchyList->SetEntryWidthRecursive\(\)](#)

sADMHierarchyList->GetExpandedIndex()

Get the expanded index

```
ASInt32 ASAPI (*GetExpandedIndex)(ADMHierarchyListRef inList,  
ADMListEntryRef inEntry);
```

Description

The **GetExpandedIndex()** returns the index of **inEntry** as though the expanded entries belonged to a non-hierarchical list.

NOTE: This function should work the same even if multiple nodes are expanded.

Parameters

inList	An ADM hierarchy list.
inEntry	An ADM list entry.

Returns

Index of **inEntry** as though the expanded entries belonged to a non-hierarchical list.

sADMHierarchyList->GetFlags()

Obtain the flags associated with the list

```
ASInt32 ASAPI (*GetFlags)(ADMHierarchyListRef inList);
```

Description

The **GetFlags()** function gets the flags associated with **inList**. A hierarchy list has an associated set of flags that control certain drawing effects associated with the list. The flags are defined in `ADMHierarchyList.h`.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

None.

See also

[sADMHierarchyList->SetFlags\(\)](#)
[sADMHierarchyList->SetFlagsRecursive\(\)](#)

sADMHierarchyList->GetGlobalLeftMargin()

Get the maximum left margin boundary

```
ASInt32 ASAPI (*GetGlobalLeftMargin)(ADMHierarchyListRef inList);
```

Description

The **GetGlobalLeftMargin()** function obtains the maximum left margin boundary for **inList**. This is where the leftmost child item of the list can begin drawing.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

None.

sADMHierarchyList->GetIndentationWidth()

Get the spacing for the expand/collapse arrow

```
ASInt32 ASAPI (*GetIndentationWidth)(ADMHierarchyListRef
inList);
```

Description

The **GetIndentationWidth()** function gets the indentation spacing for the expand/collapse arrow for **inList**.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

Indentation spacing for the expand/collapse arrow for **inList**.

See also

[sADMHierarchyList->SetIndentationWidth\(\)](#)
[sADMHierarchyList->SetIndentationWidthRecursive\(\)](#)

sADMHierarchyList->GetInitProc()

Get the ADM init function to use for hierarchy list entries

```
ADMListEntryInitProc ASAPI (*GetInitProc)(ADMHierarchyListRef
inList);
```

Description

The **GetInitProc()** function gets the initialization function being used for a hierarchy list's entries.

You probably won't call this function directly. It is called by ADM each time you call the [sADMListEntry->Create\(\)](#) function.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The initialization function used for the entries of **inList**. If you have not called [sADMHierarchyList->SetInitProc\(\)](#), returns **NULL** (not the default Init proc). Type: **ADMListEntryInitProc** (see [ADMHierarchyList.h](#))

See also

[sADMHierarchyList->SetInitProc\(\)](#)
[sADMHierarchyList->SetInitProcRecursive\(\)](#)

sADMHierarchyList->GetItem()

Get the item reference for a hierarchy list

```
ADMItemRef ASAPI (*GetItem)(ADMHierarchyListRef inList);
```

Description

The **GetItem()** function returns a reference to the ADM item to which **inList** belongs.

Since an ADM hierarchy list is an extension of a standard ADM item, you need this reference to perform standard operations on an ADM hierarchy list, such as resizing it. Once you have this reference, you can use the ADM item suite functions (see [Chapter 14, “The ADM Item Suite”](#)) to perform these operations.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The ADM item to which **inList** belongs.

sADMHierarchyList->GetLeafIndex()

Get index of entry

```
ASInt32 ASAPI (*GetLeafIndex)(ADMHierarchyListRef inList,  
ADMListEntryRef inEntry);
```

Description

The **GetLeafIndex()** function gets the index of **ADMListEntryRef** in **inList**, including child items. An entry that has no list attached to it is a leaf. If the entry has a list attached it is referred to as just an entry

Parameters

inList	An ADM hierarchy list.
ADMListEntryRef	An ADM list entry.

Returns

The index of **ADMListEntryRef** in **inList**.

sADMHierarchyList->GetLocalLeftMargin()

Get the local margin boundary

```
ASInt32 ASAPI (*GetLocalLeftMargin)(ADMHierarchyListRef
inList);
```

Description

The **GetLocalLeftMargin()** function obtains where the left margin for **inList** starts. This is where the list can begin drawing.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

Where the left margin for **inList** starts.

See also

[sADMHierarchyList->SetLocalLeftMargin\(\)](#)

sADMHierarchyList->GetLocalRect()

Get the local rectangle of a list

```
void ASAPI (*GetLocalRect)(ADMHierarchyListRef inList, ASRect*
outRect);
```

Description

The **GetLocalRect()** function gets the local rectangle of **inList**.

Parameters

inList	An ADM hierarchy list.
outRect	inList 's local rectangle. Type: outRect (see ASTypes.h)

Returns

None.

sADMHierarchyList->GetMask()

Get the filter on events received by the hierarchy list entries' tracker

```
ADMActionMask ASAPI (*GetMask)(ADMHierarchyListRef inEntry);
```

Description

The **GetMask()** function gets the mask used for controlling which events will be received by the tracker.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The action mask for **inList**. Maskable actions are listed in `ADMTracker.h`. Type: **ADMActionMask** (see `ADMTypes.h`)

See also

[sADMHierarchyList->SetMask\(\)](#)
[sADMHierarchyList->SetMaskRecursive\(\)](#)

sADMHierarchyList->GetMenuID()

Get the menu resource ID of a hierarchy list

```
ASInt32 ASAPI (*GetMenuID)(ADMHierarchyListRef inList);
```

Description

The **GetMenuID()** gets the menu resource ID of a hierarchy list. This is the standard platform menu resource used to initialize an ADM hierarchy list using [sADMHierarchyList->SetMenuID\(\)](#). If the list was initialized manually, this will be 0.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The menu resource ID of a hierarchy list.

See also

[sADMHierarchyList->SetMenuID\(\)](#)

sADMHierarchyList->GetNonLeafEntryWidth()

Get the width of a hierarchy list

```
ASInt32 ASAPI (*GetNonLeafEntryWidth)(ADMHierarchyListRef
inList;
```

Description

The **GetNonLeafEntryWidth()** function gets the width of **inList** itself.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

Width of **inList**.

sADMHierarchyList->GetNonLeafEntryTextRectRecursive()

Set the non-leaf rectangle for a hierarchy list

```
void ASAPI (*GetNonLeafEntryTextRect)(ADMHierarchyListRef
inList, ASRect* outRect);
```

Description

The **GetNonLeafEntryTextRect()** function gets the non-leaf text rectangle for **inList** and the lists of children of **inList**'s entries. This is the rectangle within which text is drawn disregarding hierarchy. Hence this is the size of the rectangle for a root level hierarchy list entry.

Parameters

inList	An ADM hierarchy list.
outRect	Non-leaf text rectangle for inList and the lists of children of inList 's entries. Type: outRect (see ASTypes.h)

Returns

None.

See also

[sADMHierarchyList->SetNonLeafEntryTextRectRecursive\(\)](#)

sADMHierarchyList->GetNotifierData()

Get the notifier data for a hierarchy list entry

```
ADMUserData ASAPI (*GetNotifierData)(ADMHierarchyListRef
inEntry);
```

Description

The **GetNotifierData()** function gets the notification data of **inList**, if any.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The notification data of **inList**. Type: **ADMUserData** (see **ADMTypes.h**)

See also

[sADMHierarchyList->SetNotifierData\(\)](#)

sADMHierarchyList->GetNotifyProc()

Get the ADM notification function being used for the hierarchy list's entries

```
ADMListEntryNotifyProc ASAPI
(*GetNotifyProc)(ADMHierarchyListRef inList);
```

Description

The **GetNotifyProc()** function gets the notification function being used for **inList**'s entries.

Rather than getting and calling an entry's notification function directly, you are more likely to use the [sADMListEntry->DefaultNotify\(\)](#) function of the ADM List Entry suite.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The notification function used for the entries of **inList**. If you have not called [sADMHierarchyList->SetNotifyProc\(\)](#), returns **NULL** (not the default Notify proc). Type: **ADMListEntryNotifyProc** (see **ADMHierarchyList.h**)

See also

[sADMHierarchyList->SetNotifyProc\(\)](#)
[sADMHierarchyList->SetNotifyProcRecursive\(\)](#)

sADMHierarchyList->GetParentEntry()

Get the parent of a selected list

```
ADMListEntryRef ASAPI (*GetParentEntry)(ADMHierarchyListRef
inList);
```

Description

The **GetParentEntry()** function returns the parent of **inList**. Returns **NULL** if **inList** is a root level hierarchy list.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

Parent of **inList** or **NULL** if **inList** is a root level hierarchy list.

sADMHierarchyList->GetTrackProc()

Get the ADM tracker function being used for the hierarchy list's entries

```
ADMListEntryTrackProc ASAPI
(*GetTrackProc)(ADMHierarchyListRef inList);
```

Description

The **GetTrackProc()** function gets the event tracking function being used for a **inList**'s entries.

Rather than getting and calling a hierarchy list's entry's tracker function directly, you are more likely to use the [sADMListEntry->DefaultTrack\(\)](#) function.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The tracker function used for the entries of **inList**. If you have not called [sADMHierarchyList->SetTrackProc\(\)](#), returns **NULL** (not the default track proc). Type: **ADMListEntryTrackProc** (see `ADMHierarchyList.h`)

See also

[sADMHierarchyList->SetTrackProc\(\)](#)
[sADMHierarchyList->SetTrackProcRecursive\(\)](#)

sADMHierarchyList->GetUserData()

Get the user data value for a hierarchy list

```
ADMUserData ASAPI (*GetUserData)(ADMHierarchyListRef inList);
```

Description

The **GetUserData()** function returns the 4-byte user value stored with the hierarchy list.

The meaning of the value is defined by the list's creator. It is likely a pointer to a data structure, for instance, the plug-in's globals. For some items, it might be a simple 4-byte type, such as a long or a fixed number.

An ADM hierarchy list's user data is independent of its item's data.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

A 4-byte value that ADM keeps with **inList**. Type: **ADMUserData** (see **ADMTypes.h**)

See also

[sADMHierarchyList->SetUserData\(\)](#)

sADMHierarchyList->GlobalToLocalPoint()

Convert a global point to list coordinates

```
void ASAPI (*GlobalToLocalPoint)(ADMHierarchyListRef inList,
ASPoint* ioPoint);
```

Description

The **GlobalToLocalPoint()** function converts a point in global coordinates to a point relative to the specified list. In this context, "global" means in the parent entry's coordinate space, so this routine is a shortcut for having to convert from the parent's coordinates to screen coordinates, and then from screen to local coordinates.

Parameters

inList	An ADM hierarchy list.
ioPoint	The point converted from "global" to local space. Type: ASPoint (see ASTypes.h)

Returns

None

See also

[sADMHierarchyList->LocalToGlobalPoint\(\)](#)
[sADMHierarchyList->GlobalToLocalRect\(\)](#)

sADMHierarchyList->GlobalToLocalRect()

Convert a global rectangle to local rectangle

```
void ASAPI (*GlobalToLocalRect)(ADMHierarchyListRef inList,
ASRect* ioRect);
```

Description

The **GlobalToLocalRect()** function converts a global rectangle to a local rectangle. In this context, “global” means in the parent entry’s coordinate space, so this routine is a shortcut for converting from the parent’s coordinates to screen coordinates, and then from screen to local coordinates.

Parameters

inList	An ADM hierarchy list.
ioRect	The rectangle converted from “global” to local space. Type: outRect (See <code>ASTypes.h</code>)

Returns

None

See also

[sADMHierarchyList->LocalToGlobalRect\(\)](#)
[sADMHierarchyList->GlobalToLocalPoint\(\)](#)

sADMHierarchyList->IndexAllSelectedEntriesInHierarchy()

Get all selected list entries

```
ADMListEntryRef ASAPI
(*IndexAllSelectedEntriesInHierarchy)(ADMHierarchyListRef
inList, ASInt32 inSelectionIndex);
```

Description

The **IndexAllSelectedEntriesInHierarchy()** function gets a reference to selected entries in **inList**, including nested and un-nested selected entries. Unlike

[sADMHierarchyList->IndexSelectedEntry\(\)](#), this routine accesses both parent entries and their children, thereby accounting for hierarchy. Use this function in conjunction with [sADMHierarchyList->NumberOfAllSelectedEntriesInHierarchy\(\)](#) to retrieve all selected entries.

Parameters

inList	An ADM hierarchy list.
inSelectionIndex	Index of the entry in inList to return a reference to.

Returns

The ADM list entry at index **inSelectionIndex**.

See also

[sADMHierarchyList->IndexSelectedEntry\(\)](#)
[sADMHierarchyList->NumberOfAllSelectedEntriesInHierarchy\(\)](#)

sADMHierarchyList->IndexEntry()

Get a list entry by index

```
ADMListEntryRef ASAPI (*IndexEntry)(ADMHierarchyListRef
inList, ASInt32 inIndex);
```

Description

The **IndexEntry()** function gets a reference to the entry at the indicated index in **inList**. Use this function with [sADMHierarchyList->NumberOfEntries\(\)](#) to iterate through all of a hierarchy list's entries, as shown in the example.

NOTE: This function only works for root level entries—children of root level entries are not searched.

Parameters

inList	An ADM hierarchy list.
inIndex	Index of the entry in inList to return a reference to.

Returns

The ADM list entry at index **inIndex**.

Example

```
ADMHierarchyListRef list;
ADMListEntryRef entry;
// assign value to list variable
ASInt32 i, count = sADMHierarchyList->NumberOfEntries(list);

for (i = 0; i < count; i++) {
    entry = sADMHierarchyList->IndexEntry(list, i);
    // do something with the entry
}
```

sADMHierarchyList->IndexExpandedEntry()

Get an expanded list entry by index

```
ADMListEntryRef ASAPI
(*IndexExpandedEntry)(ADMHierarchyListRef inList, ASInt32
inExpandedItem);
```

Description

The **IndexExpandedEntry()** function is used to get an expanded list entry by index. Use this function in conjunction with [sADMHierarchyList->NumberOfExpandedEntriesInHierarchy\(\)](#) to retrieve references to all selected entries.

Parameters

inList	An ADM hierarchy list.
inExpandedItem	Index of the expanded list entry to retrieve.

Returns

The ADM list entry at index **inExpandedItem**.

See also

[sADMHierarchyList->NumberOfExpandedEntriesInHierarchy\(\)](#)

sADMHierarchyList->IndexLeafEntry()

Get the entry at the specified index

```
ADMListEntryRef ASAPI (*IndexLeafEntry)(ADMHierarchyListRef
inList, ASInt32 inLeafItem);
```

Description

The **IndexLeafEntry()** function obtains the entry at index **inLeafItem** in **inList**, including any child items. Use this function in conjunction with

[sADMHierarchyList->NumberOfLeafEntries\(\)](#) to iterate through all leaf entries in **inList**. An entry that has no list attached to it is a leaf. If the entry has a list attached it is referred to as just an entry.

Parameters

inList	An ADM hierarchy list.
inLeafItem	Index of the leaf entry in inList to return a reference to.

Returns

The ADM list entry at index **inLeafItem**.

See also

[sADMHierarchyList->NumberOfLeafEntries\(\)](#)

sADMHierarchyList->IndexSelectedEntry()

Get one of multiple selected list entries

```
ADMListEntryRef ASAPI
(*IndexSelectedEntry)(ADMHierarchyListRef inList, ASInt32
inSelectionIndex);
```

Description

The **IndexSelectedEntry()** function gets a reference to any one of several selected entries of **inList** with multiple selections. This function only accesses entries at the root level of the hierarchy list (i.e., child entries are ignored).

Used in conjunction with the ADM Hierarchy List suite function [sADMHierarchyList->NumberOfSelectedEntries\(\)](#), you can iterate through all of a hierarchy list's selected entries (see example).

Parameters

inList	An ADM hierarchy list.
inSelectionIndex	Index of the entry in inList to return a reference to.

Returns

The ADM list entry at index **inSelectionIndex**.

Example

```
ADMHierarchyListRef list;
ADMListEntryRef entry;
ASInt32 i, count = sADMHierarchyList->NumberOfSelectedEntries( list );

for (i = 0; i < count; i++) {
    entry = sADMHierarchyList->IndexSelectedEntry(list, i);
    // do something with the selected entry
}
```

See also

[sADMHierarchyList->GetActiveEntry\(\)](#)
[sADMHierarchyList->IndexSelectedEntry\(\)](#)

sADMHierarchyList->IndexUnNestedSelectedEntriesInHierarchy()

Get all unnested selected list entries

```
ADMListEntryRef ASAPI
(*IndexUnNestedSelectedEntriesInHierarchy)(ADMHierarchyListRef
inList, ASInt32 inSelectionIndex);
```

Description

The **IndexUnNestedSelectedEntriesInHierarchy()** function is used to get references to unnested selected entries of **inList**. Children are not indexed if their parent is selected. Use this function in conjunction with [sADMHierarchyList->NumberOfUnNestedSelectedEntriesInHierarchy\(\)](#) to retrieve references to all selected entries.

Parameters

inList	An ADM hierarchy list.
inSelectionIndex	Index of the entry in inList to return a reference to.

Returns

The ADM list entry at index **inSelectionIndex**.

sADMHierarchyList->InsertEntry()

Add an entry to a hierarchy list

```
ADMListEntryRef ASAPI (*InsertEntry)(ADMHierarchyListRef
inList, ASInt32 inIndex);
```

Description

The **InsertEntry()** function adds an ADM list entry to **inList**. Zero-based **inIndex** is where the entry is to be placed within the list. The new entry is placed before entries with indices equal to or greater than **inIndex**.

This function allocates the entry and then calls your list entry init function if you have specified one. Use this function if the entry needs to be created.

Parameters

inList	An ADM hierarchy list.
inIndex	Zero-based index of where to add the new ADM list entry.

Returns

The new ADM list entry.

See also

[sADMHierarchyList->SetInitProc\(\)](#)
[sADMHierarchyList->SetInitProcRecursive\(\)](#)

sADMHierarchyList->InsertGivenEntry()

Insert entry at index point

```
ADMListEntryRef ASAPI (*InsertGivenEntry)(ADMHierarchyListRef
inList, ADMListEntryRef inEntry, ASInt32 inIndex);
```

Description

The **InsertGivenEntry()** function inserts **inEntry** at the specified index point in **inList**.

Parameters

inList	An ADM hierarchy list.
inEntry	The new ADM list entry to add.
inIndex	Index of where to add the new ADM list entry.

Returns

The newly added **inEntry**.

sADMHierarchyList->Invalidate()

Invalidate the list boundaries

```
void ASAPI (*Invalidate)(ADMHierarchyListRef inList);
```

Description

The **Invalidate()** function invalidates **inList**. This causes it to be redrawn the next time the screen is updated.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

None.

See also

[sADMHierarchyList->StartMultipleItemInvalidate\(\)](#)
[sADMHierarchyList->StopMultipleItemInvalidate\(\)](#)

sADMHierarchyList->LocalToGlobalPoint()

Convert a point to a global coordinate

```
void ASAPI (*LocalToGlobalPoint)(ADMHierarchyListRef inList,
ASPoint* ioPoint);
```

Description

The **LocalToGlobalPoint()** function converts a point in **inList** to a point in the global coordinate space. In this context, “global” means in the parent entry’s coordinate space, so this routine is a shortcut for converting local to screen coordinates, and then from screen to the parent’s local coordinates.

Parameters

inList	An ADM hierarchy list.
ioPoint	The point converted from local to “global” space. Type: ASPoint (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMHierarchyList->GlobalToLocalPoint\(\)](#)
[sADMHierarchyList->LocalToScreenPoint\(\)](#)
[sADMHierarchyList->LocalToGlobalRect\(\)](#)

sADMHierarchyList->LocalToGlobalRect()

Convert a local rectangle to global rectangle

```
void ASAPI (*LocalToGlobalRect)(ADMHierarchyListRef inList,
ASRect* ioRect);
```

Description

The **LocalToGlobalRect()** function converts a local rectangle to a global rectangle. In this context, “global” means in the parent entry’s coordinate space, so this routine is a shortcut for converting local to screen coordinates, and then from screen to the parent’s local coordinates.

Parameters

inList	An ADM hierarchy list.
ioRect	The rect converted from local to “global” space. Type: outRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMHierarchyList->GlobalToLocalRect\(\)](#)
[sADMHierarchyList->LocalToGlobalPoint\(\)](#)
[sADMHierarchyList->LocalToGlobalRect\(\)](#)

sADMHierarchyList->LocalToScreenPoint()

Convert a point to a screen coordinate

```
void ASAPI (*LocalToScreenPoint)(ADMHierarchyListRef inList,
ASPoint* ioPoint);
```

Description

The **LocalToScreenPoint()** function converts a point in **inList** to a point in the screen’s coordinate space.

Parameters

inList	An ADM hierarchy list.
ioPoint	The point in inList converted to a point in the screen coordinate space. Type: ASPoint (see ASTypes.h)

Returns

None.

See also

[sADMHierarchyList->ScreenToLocalPoint\(\)](#)
[sADMHierarchyList->LocalToGlobalPoint\(\)](#)
[sADMHierarchyList->LocalToGlobalRect\(\)](#)

sADMHierarchyList->NumberOfEntries() Get the number of list entries in a hierarchy list

```
ASInt32 ASAPI (*NumberOfEntries)(ADMHierarchyListRef inList);
```

Description

The **NumberOfEntries()** function returns the number of entries in **inList**. Used in conjunction with [sADMHierarchyList->IndexEntry\(\)](#), you can iterate through all of a list's root-level entries. Entries are not counted recursively, so the count is only for root-level entries.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The number of entries in **inList**.

See also

[sADMHierarchyList->IndexEntry\(\)](#)

sADMHierarchyList->NumberOfLeafEntries() Get the number of leaf entries

```
ASInt32 ASAPI (*NumberOfLeafEntries)(ADMHierarchyListRef inList);
```

Description

The **NumberOfLeafEntries()** function returns the number of leaf entries in **inList**, including any children items. Used in conjunction with [sADMHierarchyList->IndexLeafEntry\(\)](#), you can iterate through all of a list's leaf entries. An entry that has no list attached to it is a leaf. If the entry has a list attached it is referred to as just an entry.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The number of leaf entries in **inList**, including any children items.

See also

[sADMHierarchyList->IndexLeafEntry\(\)](#)

sADMHierarchyList->NumberOfSelectedEntries() Get the number of selected list entries in a hierarchy list

```
ASInt32 ASAPI (*NumberOfSelectedEntries)(ADMHierarchyListRef
inList);
```

Description

The **NumberOfSelectedEntries()** function returns the number of selected root-level entries in **inList**. Children are not counted if their parent is selected. Used in conjunction with the [sADMHierarchyList->IndexSelectedEntry\(\)](#), you can iterate through all selected entries of a list.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The number of selected root-level entries in **inList**.

See also

[sADMHierarchyList->IndexSelectedEntry\(\)](#)

sADMHierarchyList->NumberOfAllSelectedEntriesInHierarchy()

Get the number of all selected list entries in a hierarchy list

```
ASInt32 ASAPI (*NumberOfAllSelectedEntriesInHierarchy)
(ADMHierarchyListRef inList);
```

Description

The **NumberOfAllSelectedEntriesInHierarchy()** function returns the number of all selected entries in **inList**, including the nested ones. This routine counts entries recursively. Used in conjunction with the [sADMHierarchyList->IndexAllSelectedEntriesInHierarchy\(\)](#), you can iterate through all selected entries.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The number of all selected entries in **inList**, including the nested ones.

See also

[sADMHierarchyList->IndexAllSelectedEntriesInHierarchy\(\)](#)

sADMHierarchyList->NumberOfExpandedEntriesInHierarchy()

Get the number of expanded entries in a hierarchy list

```
ASInt32 ASAPI (*NumberOfExpandedEntries)(ADMHierarchyListRef
inList);
```

Description

The **NumberOfExpandedEntries()** function returns the number of expanded entries in **inList**. Used in conjunction with the [sADMHierarchyList->IndexExpandedEntry\(\)](#), you can iterate through all expanded entries.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The number of expanded entries in **inList**.

See also

[sADMHierarchyList->IndexExpandedEntry\(\)](#)

sADMHierarchyList->**NumberOfUnNestedSelectedEntriesInHierarchy()**

Get the number of unnested selected list entries in a hierarchy list

```
ASInt32 ASAPI (*NumberOfUnNestedSelectedEntriesInHierarchy)
(ADMHierarchyListRef inList);
```

Description

The **NumberOfUnNestedSelectedEntriesInHierarchy()** function returns the number of all selected entries in **inList** that are unnested. This routine does not recurse into selected entries, so children of selected parent entries are ignored. Used in conjunction with the [sADMHierarchyList->IndexUnNestedSelectedEntriesInHierarchy\(\)](#), you can iterate through all unnested selected entries.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

The number of all selected entries in **inList** that are unnested.

See also

[sADMHierarchyList->IndexUnNestedSelectedEntriesInHierarchy\(\)](#)

sADMHierarchyList->PickEntry()

Select a list entry at a specified point

```
ADMListEntryRef ASAPI (*PickEntry)(ADMHierarchyListRef inList,
const ASPoint* inPoint);
```

Description

This **PickEntry()** function selects the hierarchy list entry at a specified location. The location (**inPoint**) is given in the list's coordinate space.

NOTE: This function only selects root-level entries. If a point is specified that corresponds to a child entry, its root-level parent is selected.

Parameters

inList	An ADM hierarchy list.
inPoint	Location of inList entry to be selected. Type: ASPoint (see ASTypes.h)

Returns

Selected root-level entry in **inList**. If **inPoint** specifies a point that corresponds to a child entry, its root-level parent is selected and returned.

See also

[sADMHierarchyList->PickLeafEntry\(\)](#)

sADMHierarchyList->PickLeafEntry()

Select a leaf entry

```
ADMListEntryRef ASAPI (*PickLeafEntry)(ADMHierarchyListRef
inList, const ASPoint* inPoint);
```

Description

The **PickLeafEntry()** function selects the hierarchy list leaf entry at a specified location. The location (**inPoint**) is given in the list's coordinate space. This is a recursive version of [sADMHierarchyList->PickEntry\(\)](#); therefore if the specified point corresponds to a non-root entry, it is selected.

Parameters

inList	An ADM hierarchy list.
inPoint	Location of inList entry to be selected. Type: ASPoint (see ASTypes.h)

Returns

None.

See also

[sADMHierarchyList->PickEntry\(\)](#)

sADMHierarchyList->RemoveEntry()

Remove an entry from a hierarchy list

```
void ASAPI (*RemoveEntry)(ADMHierarchyListRef inList, ASInt32
inIndex);
```

Description

The **RemoveEntry()** function removes the entry at the specified index from **inList**. This function calls your list entry destroy function if you have specified one.

NOTE: This function only works for root level entries—children of root level entries are not searched.

Parameters

inList	An ADM hierarchy list.
inIndex	Index of entry to remove.

Returns

None.

sADMHierarchyList->ScreenToLocalPoint()

Convert a screen point to list coordinates

```
void ASAPI (*ScreenToLocalPoint)(ADMHierarchyListRef inList,
ASPoint* ioPoint);
```

Description

The **ScreenToLocalPoint()** function converts a point in screen coordinates to a point relative to **inList**.

Parameters

inList	An ADM hierarchy list.
ioPoint	The point in inList in screen coordinates that is converted to a point relative to inList . Type: ASPoint (see ASTypes.h)

Returns

None.

See also

[sADMHierarchyList->LocalToScreenPoint\(\)](#)
[sADMHierarchyList->LocalToGlobalPoint\(\)](#)
[sADMHierarchyList->LocalToGlobalRect\(\)](#)

sADMHierarchyList->SetBackgroundColor()

Set the background color

```
void ASAPI (*SetBackgroundColor)(ADMHierarchyListRef inList,
ADMColor inColor);
```

Description

The **SetBackgroundColorSet()** function sets the background color.

Parameters

inList	An ADM hierarchy list.
inColor	Type: ADMColor (see ADMTypes.h)

Returns

None.

sADMHierarchyList->SetDestroyProc()

Set the ADM destroy function to use for the hierarchy list's entries

```
void WINAPI (*SetDestroyProc)(ADMHierarchyListRef inList,
ADMListEntryDestroyProc inDestroyProc);
```

Description

The **SetDestroyProc()** function assigns a destroy function, **inDestroyProc**, for the entries of **inList**. This proc is called when the [sADMHierarchyList->RemoveEntry\(\)](#) function is called for a list entry.

Your function should free memory and other resources you may have allocated in your entry initialization function. ADM destroys the entry, so you do not need to call a default destroy function from within your function.

Parameters

inList	An ADM hierarchy list.
inDestroyProc	A callback with the following signature (see ADMHierarchyList.h): ADMListEntryDestroyProc(ADMListEntryRef inEntry);

Returns

None.

See also

[sADMHierarchyList->GetDestroyProc\(\)](#)
[sADMHierarchyList->SetDestroyProcRecursive\(\)](#)

sADMHierarchyList->SetDestroyProcRecursive()

Set the ADM destroy function for the

hierarchy list's entries
and children

```
void ASAPI (*SetDestroyProcRecursive)(ADMHierarchyListRef
inList, ADMListEntryDestroyProc inDestroyProc);
```

Description

The **SetDestroyProcRecursive()** function assigns a destroy function, **inDestroyProc**, for **inList** and the lists of children of the list's entries. This proc is called when the [sADMHierarchyList->RemoveEntry\(\)](#) function is called for a list entry.

Your function should free memory and other resources you may have allocated in your entry initialization function. ADM destroys the entry, so you do not need to call a default destroy function from within your function.

Parameters

inList	An ADM hierarchy list.
inDestroyProc	A callback with the following signature (see ADMHierarchyList.h): ADMListEntryDestroyProc(ADMListEntryRef inEntry);

Returns

None.

See also

[sADMHierarchyList->GetDestroyProc\(\)](#)
[sADMHierarchyList->SetDestroyProc\(\)](#)

sADMHierarchyList->SetDivided()

Make a list an ADM divided list

```
void ASAPI (*SetDivided)(ADMHierarchyListRef inList, ASBoolean
inDivided);
```

Description

The **SetDivided()** function divides **inList**. If **SetDivided()** is called on a list item, then a horizontal line is drawn between each entry. See the **Actions** palette in Illustrator as an example.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

inDivided	If true , makes inList divided; if false , inList is not divided.
------------------	---

Returns

None.

See also

[sADMHierarchyList->SetDividedRecursive\(\)](#)

sADMHierarchyList->SetDividedRecursive()

Make a list and all its children into ADM divided lists

```
void ASAPI (*SetDividedRecursive)(ADMHierarchyListRef inList,
ASBoolean inDivided);
```

Description

The **SetDividedRecursive()** function sets **inList** and the lists of children of the list's entries to the divided ADM type.

Parameters

inList	An ADM hierarchy list.
inDivided	If true , makes inList and the lists of children of the list's entries divided; if false , inList and the lists of children of the list's entries are not divided.

Returns

None.

See also

[sADMHierarchyList->SetDivided\(\)](#)

sADMHierarchyList->SetDrawProc()

Set the ADM drawing function to use for the hierarchy list's entries

```
void ASAPI (*SetDrawProc)(ADMHierarchyListRef inList,
ADMListEntryDrawProc inDrawProc);
```

Description

The **SetDrawProc()** function defines a drawing function, **inDrawProc**, for the entries of **inList**.

To call the default draw function for an entry, you need to use the [sADMLEntry->DefaultDraw\(\)](#) function.

See [Using Event Callbacks](#) in [Chapter 1, “ADM Overview”](#) for more information.

Parameters

inList	An ADM hierarchy list.
inDrawProc	<p>A callback with the following signature (see <code>ADMHierarchyList.h</code>):</p> <pre>ADMLEntryDrawProc(ADMLEntryRef inEntry, ADMDrawerRef inDrawer);</pre> <p>Within your draw function you can use the ADM Drawer suite functions to do standard image operations such as drawing lines and pictures. The inDrawer argument is passed to the ADM Drawer suite functions to indicate where the imaging is to occur.</p>

Returns

None.

See also

[sADMHierarchyList->GetDrawProc\(\)](#)
[sADMHierarchyList->SetDrawProcRecursive\(\)](#)

sADMHierarchyList->SetDrawProcRecursive()

Set the ADM drawing function for the hierarchy list's entries and any children

```
void ASAPI (*SetDrawProcRecursive)(ADMHierarchyListRef inList,  
ADMLEntryDrawProc inDrawProc);
```

Description

The **SetDrawProcRecursive()** function defines a drawing function, **inDrawProc**, for **inList** and the lists of children of the list's entries.

To call the default draw function for an entry, you need to use the [sADMLEntry->DefaultDraw\(\)](#) function.

See [Using Event Callbacks](#) in [Chapter 1, “ADM Overview”](#) for more information.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

inDrawProc	<p>A callback with the following signature (see <code>ADMHierarchyList.h</code>):</p> <pre>ADMListEntryDrawProc(ADMListEntryRef inEntry, ADMDrawerRef inDrawer);</pre> <p>Within your draw function you can use the ADM Drawer suite functions to do standard image operations such as drawing lines and pictures. The inDrawer argument is passed to the ADM Drawer suite functions to indicate where the imaging is to occur.</p>
-------------------	---

Returns

None.

See also

[sADMHierarchyList->GetDrawProc\(\)](#)
[sADMHierarchyList->SetDrawProcRecursive\(\)](#)

sADMHierarchyList->SetEntryHeight()

Set the height of a hierarchy list's entries

```
void ASAPI (*SetEntryHeight)(ADMHierarchyListRef inList,  
ASInt32 inHeight);
```

Description

This **SetEntryHeight()** function sets the height of **inList**'s entries and, indirectly, the number of rows that appear in the list. All entries have the same height.

Parameters

inList	An ADM hierarchy list.
inHeight	The height of a hierarchy list's entry.

Returns

None.

See also

[sADMHierarchyList->GetEntryHeight\(\)](#)
[sADMHierarchyList->SetEntryHeightRecursive\(\)](#)

sADMHierarchyList->SetEntryHeightRecursive()

Set the height of a hierarchy list's entries and any children

```
void ASAPI (*SetEntryHeightRecursive)(ADMHierarchyListRef
inList, ASInt32 inHeight);
```

Description

The **SetEntryHeightRecursive()** function sets the height of **inList**'s entries and any children and indirectly the number of rows that will appear in the list. All entries have the same height.

Parameters

inList	An ADM hierarchy list.
inHeight	The height of a hierarchy list's entry.

Returns

None.

See also

[sADMHierarchyList->GetEntryHeight\(\)](#)
[sADMHierarchyList->SetEntryHeight\(\)](#)

sADMHierarchyList->SetEntryTextRect()

Set the edit text rectangle for a hierarchy list

```
void ASAPI (*SetEntryTextRect)(ADMHierarchyListRef inList,
const ASRect* inRect);
```

Description

The **SetEntryTextRect()** function sets the edit-in-place text item of a hierarchy list.

Parameters

inList	An ADM hierarchy list.
inRect	Location of the edit-in-place text item of inList . Type: outRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMHierarchyList->GetEntryTextRect\(\)](#)
[sADMHierarchyList->SetEntryTextRectRecursive\(\)](#)

sADMHierarchyList->SetEntryTextRectRecursive()

Recursively set the edit text rectangle for a hierarchy list.

```
void ASAPI (*SetEntryTextRectRecursive)(ADMHierarchyListRef
inList, const ASRect* inRect);
```

Description

The **SetEntryTextRectRecursive()** function recursively sets the edit-in-place text items of **inList**'s entries and children.

Parameters

inList	An ADM hierarchy list.
inRect	Location of the edit-in-place text items of inList and any children. Type: outRect (see ASTypes.h)

Returns

None.

See also

[sADMHierarchyList->GetEntryTextRect\(\)](#)
[sADMHierarchyList->SetEntryTextRect\(\)](#)

sADMHierarchyList->SetEntryWidth()

Set the width of a column of list entries

```
void ASAPI (*SetEntryWidth)(ADMHierarchyListRef inList,
ASInt32 inWidth);
```

Description

The **SetEntryWidth()** function sets the width of the list entries' local and bounds rect.

If the list is part of a listbox item and has the style **kADMTileListBoxStyle**, this function sets the width of a column. For listbox items of style **kADMTileListBoxStyle**, the listbox is made up of rows and columns. This function sets the width of a column and indirectly sets the number of columns to appear. It will not create rows for other list types and styles, but may affect the list's appearance.

Parameters

inList	An ADM hierarchy list.
inWidth	Width of a column if inList is part of a listbox item and has the style kADMTileListBoxStyle ; for any other list style, is the width of the single column of the list.

Returns

None.

See also

[sADMHierarchyList->GetEntryWidth\(\)](#)
[sADMHierarchyList->SetEntryWidthRecursive\(\)](#)

sADMHierarchyList->SetEntryWidthRecursive()

Set the width of a column of list entries and any children

```
void ASAPI (*SetEntryWidthRecursive)(ADMHierarchyListRef
inList, ASInt32 inWidth);
```

Description

The **SetEntryWidthRecursive()** function sets the width of the list entries' local and bounds rect, including children.

If the list is part of a listbox item and has the style **kADMTileListBoxStyle**, this function sets the width of a column. For listbox items of style **kADMTileListBoxStyle**, the listbox is made up of rows and columns. This function sets the width of a column and indirectly sets the number of columns to appear. It will not create rows for other list types and styles, but may affect the list's appearance.

Parameters

inList	An ADM hierarchy list.
inWidth	Width of a column if inList is part of a listbox item and has the style kADMTileListBoxStyle ; for any other list style, is the width of the single column of the list, including children.

Returns

None.

See also

[sADMHierarchyList->GetEntryWidth\(\)](#)
[sADMHierarchyList->SetEntryWidth\(\)](#)

sADMHierarchyList->SetFlags()

Set the flags associated with the list

```
void ASAPI (*SetFlags)(ADMHierarchyListRef inList, ASInt32
inFlags);
```

Description

The **SetFlags()** function sets the flags associated with **inList**. A hierarchy list has an associated set of flags that control certain drawing effects associated with the list. The flags are defined in `ADMHierarchyList.h`.

```
// Flags
#define kMultiSelectAcrossHierarchyLevels      0x00000001
#define kHierarchyListEndedWithFrame          0x00000002
#define kHierarchyListTopLevelSpecialBackground 0x00000004
#define kHierarchyListLeafOnly                 0x00000008
```

kMultiSelectAcrossHierarchyLevels enables selection across hierarchy levels. **kHierarchyListEndedWithFrame** draws a black line at the bottom of the list. **kHierarchyListTopLevelSpecialBackground** draws a gray background. **kHierarchyListLeafOnly** indicates that entries cannot have sublists (applies to root level list entries). In this latter case, the hierarchy list is used primarily for organization.

Parameters

inList	An ADM hierarchy list.
inFlags	Sets flags. Values are defined in <code>ADMHierarchyList.h</code> .

Returns

None.

See also

[sADMHierarchyList->GetFlags\(\)](#)
[sADMHierarchyList->SetFlagsRecursive\(\)](#)

sADMHierarchyList->SetFlagsRecursive()Set the flags associated with
the list and its children

```
void ASAPI (*SetFlagsRecursive)(ADMHierarchyListRef inList,
ASInt32 inFlags);
```

Description

The **SetFlagsRecursive()** function sets the flags associated with **inList** and the lists of children of the list's entries. A hierarchy list has an associated set of flags

that control certain drawing effects associated with the list. The flags are defined in `ADMHierarchyList.h`.

```
// Flags
#define kMultiSelectAcrossHierarchyLevels    0x00000001
#define kHierarchyListEndedWithFrame        0x00000002
#define kHierarchyListTopLevelSpecialBackground 0x00000004
#define kHierarchyListLeafOnly              0x00000008
```

kMultiSelectAcrossHierarchyLevels enables selection across hierarchy levels. **kHierarchyListEndedWithFrame** draws a black line at the bottom of the list. **kHierarchyListTopLevelSpecialBackground** draws a gray background. **kHierarchyListLeafOnly** indicates that entries cannot have sublists (applies to root level list entries). In this latter case, the hierarchy list is used primarily for organization.

Parameters

inList	An ADM hierarchy list.
inFlags	Sets flags. Values are defined in <code>ADMHierarchyList.h</code> .

Returns

None.

See also

[sADMHierarchyList->GetFlags\(\)](#)
[sADMHierarchyList->SetFlags\(\)](#)

sADMHierarchyList->SetIndentationWidth()

Set the spacing for the expand/collapse arrow

```
void ASAPI (*SetIndentationWidth)(ADMHierarchyListRef inList,
ASInt32 inWidth);
```

Description

The **SetIndentationWidth()** function sets the indentation spacing for the expand/collapse arrow for **inList**.

Parameters

inList	An ADM hierarchy list.
inWidth	Indentation spacing for the expand/collapse arrow for inList .

Returns

None.

See also

[sADMHierarchyList->GetIndentationWidth\(\)](#)
[sADMHierarchyList->SetIndentationWidthRecursive\(\)](#)

sADMHierarchyList->SetIndentationWidthRecursive()

Set the spacing for the expand/collapse arrow for a list and its children

```
void WINAPI (*SetIndentationWidthRecursive)(ADMHierarchyListRef
inList, ASInt32 inWidth);
```

Description

The **SetIndentationWidth()** function sets the indentation spacing for the expand/collapse arrow for **inList** and the lists of children of the list's entries.

Parameters

inList	An ADM hierarchy list.
inWidth	Indentation spacing for the expand/collapse arrow for inList .

Returns

None.

See also

[sADMHierarchyList->GetIndentationWidth\(\)](#)
[sADMHierarchyList->SetIndentationWidth\(\)](#)

sADMHierarchyList->SetInitProc()

Set the ADM init function to use for hierarchy list entries

```
void WINAPI (*SetInitProc)(ADMHierarchyListRef inList,
ADMListEntryInitProc inInitProc);
```

Description

The **SetInitProc()** function defines an initialization proc, **inInitProc**, for **inList**. This function sets the initialization proc for the root level list items, but does not affect the items in sub-lists of the specified list. This will be called each time an entry is created within the indicated list.

Within your init function you can allocate memory or other resources and you can initialize variables. You do not need to allocate the entry itself, as ADM handles this.

NOTE: **inList** does not have an initialization function. This is handled at the ADM Item object level.

Parameters

inList	An ADM hierarchy list.
inInitProc	A callback with the following signature (see <code>ADMHierarchyList.h</code>): ADMListEntryInitProc(ADMListEntryRef inEntry); Within this callback you can allocate memory or other resources and you can initialize variables. You do not need to allocate the entry itself, as ADM handles this.

Returns

None.

See also

[sADMHierarchyList->GetInitProc\(\)](#)
[sADMHierarchyList->SetInitProcRecursive\(\)](#)

sADMHierarchyList->SetInitProcRecursive()

Set the ADM init function to use for hierarchy list entries and their children

```
void ASAPI (*SetInitProcRecursive)(ADMHierarchyListRef inList,
ADMListEntryInitProc inInitProc)
```

Description

The **SetInitProcRecursive()** function defines an initialization proc, **inInitProc**, for **inList** and the lists of children of **inList**'s entries. This function sets the initialization proc for the root level list items, but does not affect the items in sub-lists of the specified list. This will be called each time an entry is created within the indicated list.

Within your init function you can allocate memory or other resources and you can initialize variables. You do not need to allocate the entry itself, as ADM handles this.

NOTE: **inList** does not have an initialization function. This is handled at the ADM Item object level.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

inInitProc A callback with the following signature (see `ADMHierarchyList.h`):

```
ADMListEntryInitProc(ADMListEntryRef inEntry);
```

Within this callback you can allocate memory or other resources and you can initialize variables. You do not need to allocate the entry itself, as ADM handles this.

Returns

None.

See also

[sADMHierarchyList->GetInitProc\(\)](#)
[sADMHierarchyList->SetInitProc\(\)](#)

sADMHierarchyList->SetLocalLeftMargin()

Set where a list can draw
 (the local left margin
 boundary)

```
void ASAPI (*SetLocalLeftMargin)(ADMHierarchyListRef inList,  
ASInt32 inWidth);
```

Description

The **SetLocalLeftMargin()** function sets the left margin for **inList**. This is where the list can begin drawing.

Parameters

inList	An ADM hierarchy list.
inWidth	Width of left margin, in pixels.

Returns

None.

See also

[sADMHierarchyList->GetLocalLeftMargin\(\)](#)

sADMHierarchyList->SetMask()

Set a filter on events received by the
 hierarchy list entries' tracker

```
void ASAPI (*SetMask)(ADMHierarchyListRef inEntry,  
ADMActionMask inMask);
```

Description

The **SetMask()** function sets the mask for controlling which events are received by the tracker for **inList**.

Parameters

inList	An ADM hierarchy list.
inMask	The action mask for inList . Maskable actions are listed in <code>ADMTracker.h</code> . Type: ADMActionMask (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMHierarchyList->GetMask\(\)](#)
[sADMHierarchyList->SetMaskRecursive\(\)](#)

sADMHierarchyList->SetMaskRecursive()

Set a filter on events received by the hierarchy list entries' tracker

```
void ASAPI (*SetMaskRecursive)(ADMHierarchyListRef inEntry,
    ADMActionMask inMask);
```

Description

The **SetMaskRecursive()** function sets the mask for controlling which events are received by the tracker for **inList** and the lists of children of the list's entries.

Parameters

inList	An ADM hierarchy list.
inMask	The action mask for inList . Maskable actions are listed in <code>ADMTracker.h</code> . Type: ADMActionMask (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMHierarchyList->GetMask\(\)](#)
[sADMHierarchyList->SetMask\(\)](#)

sADMHierarchyList->SetMenuID()

Set the menu resource ID of a hierarchy list

```
void ASAPI (*SetMenuID)(ADMHierarchyListRef inList,
SPPluginRef inMenuResPlugin, ASInt32 inMenuResID, const char*
inMenuResName);
```

Description

The **SetMenuID()** function sets the menu resource ID of **inList**. Setting the menu ID causes ADM to read the resource and map the platform menu items to ADM hierarchy list entries. **inMenuResID** is an ID of a standard platform menu resource.

If the list has already been assigned a resource ID and **SetMenuID()** is called again, the existing list will be disposed before the new one is added.

Parameters

inList	An ADM hierarchy list.
inMenuResPlugin	Plug-in reference.
inMenuResID	Resource ID for inList . ID of standard platform menu resource.
inMenuResName	List resource name.

Returns

None.

See also

[sADMHierarchyList->GetMenuID\(\)](#)

sADMHierarchyList->SetNonLeafEntryTextRect()

Set the non-leaf rectangle for a hierarchy list

```
void ASAPI (*SetNonLeafEntryTextRect)(ADMHierarchyListRef
inList, const ASRect* inRect);
```

Description

The **SetNonLeafEntryTextRect()** function sets the non-leaf text rectangle of a hierarchy list. This is the rectangle within which text is drawn disregarding hierarchy. Hence this is the size of the rectangle for a root level hierarchy list entry.

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Non-leaf text rectangle of **inList**. Type: **outRect** (see `ASTypes.h`)

Returns

None.

See also

[sADMHierarchyList->GetNonLeafEntryTextRectRecursive\(\)](#)

sADMHierarchyList->SetNonLeafEntryTextRectRecursive()

Set the non-leaf rectangle for a hierarchy list and any children

```
void WINAPI
(*SetNonLeafEntryTextRectRecursive)(ADMHierarchyListRef
inList, const ASRect* inRect);
```

Description

The **SetNonLeafEntryTextRectRecursive()** function sets the non-leaf text rectangle for **inList** and the lists of children of **inList**'s entries.

Parameters

inList	An ADM hierarchy list.
inRect	The non-leaf text rectangle for inList and the lists of children of inList 's entries. Type: outRect (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMHierarchyList->GetNonLeafEntryWidth\(\)](#)

sADMHierarchyList->SetNotifierData()

Set notifier data for a hierarchy list entry

```
void WINAPI (*SetNotifierData)(ADMHierarchyListRef inEntry,
ADMUserData inData);
```

Description

The **SetNotifierData()** function sets the notification data for **inList**, if any. This is used for custom notification procedures.

Parameters

inList	An ADM hierarchy list.
inData	Custom notification data. Type: ADMUserData (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMHierarchyList->GetNotifierData\(\)](#)

sADMHierarchyList->SetNotifyProc()

Set the ADM notification function to use for the hierarchy list's entries

```
void ASAPI (*SetNotifyProc)(ADMHierarchyListRef inList,
ADMListEntryNotifyProc inNotifyProc);
```

Description

The **SetNotifyProc()** function assigns an event notification function, **inNotifyProc**, to the entries of **inList**. The notification is sent after an event (e.g., mouse-up) occurs on the entry.

Within your notification function you can use the ADM Notifier suite functions to determine the type of notification received. The **inNotifier** argument is passed to the ADM Notifier suite functions to indicate the event for which information is being requested.

See [Using Event Callbacks](#) in [Chapter 1, "ADM Overview"](#) for an example.

Parameters

inList	An ADM hierarchy list.
inNotifyProc	A callback with the following signature (see <code>ADMHierarchyList.h</code>): ADMListEntryNotifyProc (ADMListEntryRef inEntry, ADMNotifierRef inNotifier); Use inNotifier to use the functions in the ADM Notifier suite. You can use the ADM Notifier suite functions to determine the type of notification received. The inNotifier argument is passed to the ADM Notifier suite functions to indicate the event for which information is being requested.

Returns

None.

See also

[sADMHierarchyList->SetNotifyProcRecursive\(\)](#)
[sADMHierarchyList->GetNotifyProc\(\)](#)

sADMHierarchyList->SetNotifyProcRecursive()

Set the ADM notification function for the hierarchy list's entries and children

```
void WINAPI (*SetNotifyProcRecursive)(ADMHierarchyListRef  
inList, ADMListEntryNotifyProc inNotifyProc);
```

Description

The **SetNotifyProcRecursive()** function assigns an event notification function, **inNotifyProc**, for **inList** and the lists of children of the **inList**'s entries. The notification is sent after an event (e.g., mouse-up) occurs on the entry.

Within your notification function you can use the ADM Notifier suite functions to determine the type of notification received. The **inNotifier** argument is passed to the ADM Notifier suite functions to indicate the event for which information is being requested.

See [Using Event Callbacks](#) in [Chapter 1, "ADM Overview"](#) for an example.

Parameters

inList	An ADM hierarchy list.
inNotifyProc	<p>A callback with the following signature (see <code>ADMHierarchyList.h</code>):</p> <pre>ADMListEntryNotifyProc (ADMListEntryRef inEntry, ADMNotifierRef inNotifier);</pre> <p>Use inNotifier to use the functions in the ADM Notifier suite. You can use the ADM Notifier suite functions to determine the type of notification received. The inNotifier argument is passed to the ADM Notifier suite functions to indicate the event for which information is being requested.</p>

Returns

None.

See also

[sADMHierarchyList->SetNotifyProcRecursive\(\)](#)
[sADMHierarchyList->GetNotifyProc\(\)](#)

sADMHierarchyList->SetTrackProc()

Set the ADM tracker function to use for the hierarchy list's entries

```
void WINAPI (*SetTrackProc)(ADMHierarchyListRef inList,
ADMListEntryTrackProc inTrackProc);
```

Description

The **SetTrackProc()** function defines an event tracking callback, **inTrackProc**, for the entries of **inList**.

Within your tracker function you can use the ADM Tracker suite functions to access event information. The **inTracker** argument is passed to the ADM Tracker suite functions to indicate the event for which information is being requested.

See [Using Event Callbacks](#) in [Chapter 1, “ADM Overview”](#) for an example.

Parameters

inList	An ADM hierarchy list.
inTrackProc	<p>A callback with the following signature (see <code>ADMHierarchyList.h</code>):</p> <pre>ASBoolean ADMListEntryTrackProc (ADMListEntryRef inEntry, ADMTrackerRef inTracker);</pre> <p>inTracker is passed to the ADM Tracker suite functions to indicate the event for which information is being requested. This callback returns a boolean—if true, the item receives a notify event when the mouse is released; if false, a notify event will not be received. It always means whether or not to send notification <i>except</i> for keystroke events. In those cases it means whether or not the keystroke was handled.</p>

Returns

None.

See also

[sADMHierarchyList->GetTrackProc\(\)](#)
[sADMHierarchyList->SetTrackProcRecursive\(\)](#)

sADMHierarchyList->SetTrackProcRecursive()

Set the ADM tracker function to use for the hierarchy list's entries, including children

```
void WINAPI (*SetTrackProcRecursive)(ADMHierarchyListRef
inList, ADMListEntryTrackProc inTrackProc);
```

Description

The **SetTrackProcRecursive()** function defines a recursive event tracking function, **inTrackProc**, for **inList** and the lists of children of **inList**'s entries.

Within your tracker function you can use the ADM Tracker suite functions to access event information. The **inTracker** argument is passed to the ADM Tracker suite functions to indicate the event for which information is being requested.

See [Using Event Callbacks](#) in [Chapter 1, "ADM Overview"](#) for an example.

Parameters

inList	An ADM hierarchy list.
inTrackProc	A callback with the following signature (see <code>ADMHierarchyList.h</code>): <pre>ADMListEntryTrackProc (ADMListEntryRef inEntry, ADMTTrackerRef inTracker);</pre> inTracker is passed to the ADM Tracker suite functions to indicate the event for which information is being requested. This callback returns a boolean—if true , the item receives a notify event when the mouse is released; if false , a notify event will not be received. It always means whether or not to send notification <i>except</i> for keystroke events. In those cases it means whether or not the keystroke was handled.

Returns

None.

See also

[sADMHierarchyList->GetTrackProc\(\)](#)
[sADMHierarchyList->SetTrackProc\(\)](#)

sADMHierarchyList->SetUserData()

Set the user data value for a hierarchy list

```
void WINAPI (*SetUserData)(ADMHierarchyListRef inList,
ADMUserData inData);
```


Description

The **SetUserData()** function sets the 4-byte user value stored with the hierarchy list.

An ADM hierarchy list's user data is independent of its items' data. To set an item's user data, get the item from the list and then get that item's user data using the list suite functions.

Parameters

inList	An ADM hierarchy list.
inData	A 4-byte value that ADM keeps with the list; you can pass a pointer to a block of memory or some other value and retrieve it later using the sADMHierarchyList->GetUserData() function. Type: ADMUserData (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMHierarchyList->GetUserData\(\)](#)

sADMHierarchyList->StartMultipleItemInvalidate()

Start invalidation.

```
void ASAPI (*StartMultipleItemInvalidate)(ADMHierarchyListRef
inList);
```

Description

The **StartMultipleItemInvalidate()** function allows you to limit invalidation to improve performance. Use with [sADMHierarchyList->StopMultipleItemInvalidate\(\)](#).

Parameters

inList	An ADM hierarchy list.
---------------	------------------------

Returns

None.

See also

[sADMHierarchyList->StopMultipleItemInvalidate\(\)](#)
[sADMHierarchyList->Invalidate\(\)](#)

sADMHierarchyList->StopMultipleItemInvalidate()

Stop invalidation

```
void ASAPI (*StopMultipleItemInvalidate)(ADMHierarchyListRef
inList);
```

Description

The **StartMultipleItemInvalidate()** function allows you to limit invalidation to improve performance. Use with [sADMHierarchyList->StartMultipleItemInvalidate\(\)](#).

Parameters

inList	An ADM hierarchy list.
inFromIndex	Entry to be swapper with inToIndex .
inToIndex	Entry to be swapper with inFromIndex .

Returns

None.

See also

[sADMHierarchyList->StartMultipleItemInvalidate\(\)](#)

[sADMHierarchyList->Invalidate\(\)](#)

sADMHierarchyList->SwapEntries()

Switch position between two entries

```
void ASAPI (*SwapEntries)(ADMHierarchyListRef inList, ASInt32
inFromIndex, ASInt32 inToIndex);
```

Description

The **SwapEntries()** function switches the position of two entries.

Parameters

inList	An ADM hierarchy list.
inFromIndex	Entry to be swapper with inToIndex .
inToIndex	Entry to be swapper with inFromIndex .

Returns

None.

sADMHierarchyList->UnlinkEntry()

Unlink an entry from a hierarchy list

```
ADMListEntryRef ASAPI (*UnlinkEntry)(ADMHierarchyListRef
inList, ASInt32 inIndex);
```

Description

The **UnlinkEntry()** function unlinks the entry at the specified index from **inList**. To “unlink” the entry means remove the specified entry from its list; the entry is then returned. This can be used to implement drag-and-drop.

Parameters

inList	An ADM hierarchy list.
inIndex	Index of the list entry to be unlinked.

Returns

Unlinked ADM list entry at **inIndex**.

12

The ADM Icon Suite

About the ADM Icon Suite

The ADM Icon suite provides a standard interface to picture resources on multiple platforms. The suite currently supports pictures and icons on Macintosh and Windows platforms and refers to them as ADM icons.

Accessing the Suite

The ADM Icon suite is referred to as:

```
#define kADMIconSuite          "ADM Icon Suite"
```

with the version constant:

```
#define kADMIconSuiteVersion2      2
```

NOTE: Determine the suite version number you are using by examining the `ADMIcon.h` header file.

The suite is acquired as follows:

```
ADMIconSuite *sADMIcon;  
error = SSPBasic->AcquireSuite(kADMIconSuite, kADMIconSuiteVersion2,  
    &sADMIcon);  
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

ADM Icons

ADM provides a generic icon interface to a number of platform-based resource types. When an icon is read from a plug-in file, ADM searches the supported resource types for the given resource ID. The resource is read into memory and a reference to the ADM icon is returned to the caller.

The supported resource types are (see `ADMIcon.h`):

```
typedef enum {  
    // Mac types  
    kCICN, kPICT, kIconSuite,  
    // Windows types  
    kWinIcon, kBMP,  
    // Either type  
    kADMImageIcon,  
    kUnknown  
} ADMIconType;
```

The search order for the resources is the enumeration order shown. If the resource ID exists in more than one type, the first one found is returned. This includes the icon suite types. **kUnknown** indicates that the icon could not be found in the file; icon drawing routines will not draw anything for icons of this type.

Icons of type **kIconSuite** and **kWinIcon** can have multiple icons with multiple depths, but all the supplied icons should have the same dimensions. On Macintosh, icon suite resources are searched in the order:

Large (ICN#/icl4/icl8)

Small (ics#/ics4/ics8)

Mini (icm#/ics4/ics8)

NOTE: On Macintosh, the **CICN** resource is provided for backward compatibility with Adobe Illustrator 6.0. Resources of this type are not a part of the resource search. Icon suites are the preferred format.

To draw an ADM icon, use the functions [sADMDrawer->DrawIcon\(\)](#) and [sADMDrawer->DrawIconCentered\(\)](#).

ADM Icon Suite Functions

sADMIcon->Create()

Create a new icon

```
ADMIconRef ASAPI (*Create)(ADMIconType inIconType, ASInt32
inWidth, ASInt32 inHeight, void* inData);
```

Description

The **Create()** function creates a new icon of type **inIconType** using the provided information. The function returns a reference to the new icon.

It is the caller's responsibility to provide the data in a valid format. Families of icons should all be the same size.

ADM does not make a copy of the icon data. If you release the memory, be sure to call [sADMIcon->Destroy\(\)](#) to inform ADM that it is no longer valid. This includes static memory in a plug-in that becomes invalid if the plug-in is unloaded.

Parameters

inIconType	Icon type. Type: ADMIconType (see <code>ADMIcon.h</code>):
inWidth	Width of icon (in pixels).
inHeight	Height of icon (in pixels).

inData	A 4-byte value that ADM keeps with the icon; you can pass a pointer to a block of memory or some other value and retrieve it later using the sADMIcon->GetData() function.
---------------	---

Returns

The new icon.

See also

[sADMIcon->Destroy\(\)](#)
[sADMIcon->GetData\(\)](#)

sADMIcon->CreateFromImage()

Take ownership of an image

```
ADMIconRef ASAPI (*CreateFromImage)(ADMImageRef inImage);
```

Description

The **CreateFromImage()** function enables ADM icon to take ownership of the image.

Parameters

inImage	An ADM image.
----------------	---------------

Returns

A new ADM icon that now owns **inImage**.

See also

[sADMIcon->Create\(\)](#)
[sADMIcon->Destroy\(\)](#)

sADMIcon->Destroy()

Remove an icon from memory

```
void ASAPI (*Destroy)(ADMIconRef inIcon);
```

Description

The **Destroy()** function removes an existing icon from memory. If the icon was created using the [sADMIcon->GetFromResource\(\)](#) function, **Destroy()** frees the image data. If the icon was created with the [sADMIcon->Create\(\)](#) function, the caller should free the image memory before calling **Destroy()**.

Parameters

inIcon	An ADM icon.
---------------	--------------

Returns

None.

See also

[sADMIcon->Create\(\)](#)
[sADMIcon->GetFromResource\(\)](#)

sADMIcon->GetData()

Get the pixel data of an existing icon

```
ASAPI void* (*GetData)(ADMIconRef inIcon);
```

Description

The **GetData()** function gets the image data for **inIcon**. The returned value is a reference to the image data.

If you need the actual pixel information, you must parse the image data of the icon data reference according to the platform resource specification and/or use OS-specific functions.

Parameters

inIcon	An ADM icon.
---------------	--------------

Returns

A pointer to the 4-byte value that ADM keeps with the icon that was set using [sADMIcon->Create\(\)](#).

See also

[sADMIcon->Create\(\)](#)

sADMIcon->GetFromResource()

Read an icon from a resource

```
ADMIconRef ASAPI (*GetFromResource)(SPPluginRef inPluginRef,  
const char* inName, ASInt32 inIconID, ASInt32 inIconIndex);
```

Description

The **GetFromResource()** function gets an icon resource from the plug-in file. The plug-in reference is passed in the **SPMessageData** data structure when the plug-in is loaded. **inIconID** and **inIconIndex** are added together to get the resource ID of the icon to be loaded. The icon will be one of the standard ADM icon types (see [sADMIcon->Create\(\)](#)).

To determine the type or size of the icon, use the ADM Icon suite information functions. The function [sADMIcon->GetType\(\)](#) will return **kUnknown** if the icon could not be found.

NOTE: The search order of resources is PICT/BMP followed by IconSuite/Icon. On Macintosh, this function will not use resources of type 'CICN' even if they are available.

Parameters

inPluginRef	Plug-in reference.
inName	Internal identifier for the icon. Must be unique.
inIconID	Platform native resource ID. This ID refers to the platform specific code provided to ADM to create the icon on either the Macintosh or PC.
inIconIndex	Index added to inIconID to get the final resource ID of the icon.

Returns

An ADM icon. Type: **ADMIconRef** (see `ADMTypes.h`)

See also

[sADMIcon->Create\(\)](#)
[sADMIcon->GetType\(\)](#)

sADMIcon->GetHeight()

Get the height of an existing icon

```
ASInt32 ASAPI (*GetHeight)(ADMIconRef inIcon);
```

Description

The **GetHeight()** function gets height information for **inIcon**. The height gives the vertical dimension of the icon in pixels.

Parameters

inIcon	An ADM icon.
---------------	--------------

Returns

The vertical dimension of **inIcon** (in pixels).

See also

[sADMIcon->GetWidth\(\)](#)

sADMIcon->GetType()

Get the resource type of an icon

```
ADMIconType ASAPI (*GetType)(ADMIconRef inIcon);
```

Description

The **GetType()** function gets type information for **inIcon**. The icon type is returned and indicates the resource type of the icon as it was read from the file.

After calling the [sADMIcon->GetFromResource\(\)](#) function, the type can be checked. It will be **kUnknown** if the icon could not be found.

Parameters

inIcon	An ADM icon.
---------------	--------------

Returns

The icon type. Type: **ADMIconType** (see `ADMIcon.h`)

See also

[sADMIcon->GetFromResource\(\)](#)

sADMIcon->GetWidth()

Get the width of an existing icon

```
ASInt32 ASAPI (*GetWidth)(ADMIconRef inIcon);
```

Description

The **GetWidth()** function gets width information for **inIcon**. The width is the horizontal dimension of the icon in pixels.

Parameters

inIcon	An ADM icon.
---------------	--------------

Returns

The horizontal dimension of **inIcon** (in pixels).

See also

[sADMIcon->GetHeight\(\)](#)

sADMIcon->IsFromResource()

Find out whether an icon is from a resource

```
ASBoolean ASAPI (*IsFromResource)(ADMIconRef inIcon);
```

Description

The **IsFromResource()** function gets information about how **inIcon** was created. It returns **true** if **inIcon** was read from a plug-in resource file; it returns **false** if **inIcon** was made using the [sADMIcon->Create\(\)](#) function.

Parameters

inIcon	An ADM icon.
---------------	--------------

Returns

true if **inIcon** was read from a plug-in resource file; **false** if **inIcon** was made using the [sADMIcon->Create\(\)](#) function.

See also

[sADMIcon->Create\(\)](#)

13

The ADM Image Suite

About the ADM Image Suite

The ADM Image suite provides a means for creating off-screen images that can be displayed and manipulated with ADM Drawer suite functions (see [Chapter 9, “The ADM Drawer Suite”](#)).

Accessing the Suite

The ADM Image suite is referred to as:

```
#define kADMImageSuite                "ADM Image Suite"
```

with the version constant:

```
#define kADMImageSuiteVersion2        2
```

NOTE: Determine the suite version number you are using by examining the `ADMImage.h` header file.

The suite is acquired as follows:

```
ADMImageSuite *sADMImage;
error = SSPBasic->AcquireSuite(kADMImageSuite, kADMImageSuiteVersion2,
    &sADMImage);
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

ADM Image Suite Functions

sADMImage->BeginADMDrawer()

Begin using ADM Drawer suite functions to draw an image

```
ADMDrawerRef ASAPI (*BeginADMDrawer)(ADMImageRef inImage);
```

Description

The **BeginADMDrawer()** function enables use of the ADM Drawer suite functions to draw **inImage**. When finished using ADM Drawer suite operations on the image, it should be released with [sADMImage->EndADMDrawer\(\)](#).

Parameters

inImage	An ADM image.
----------------	---------------

Returns

An ADM drawer.

See also

[sADMImage->EndADMDrawer\(\)](#)

sADMImage->BeginAGMImageAccess()

Begin using AGM routines to draw an image

```
void ASAPI (*BeginAGMImageAccess)(ADMImageRef inImage, struct
_t_ADMAGMImageRecord* outImageRecord);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **BeginAGMImageAccess()** function starts drawing **inImage** with AGM routines. When you are finished using AGM routines on the image, it should be released with [sADMImage->EndAGMImageAccess\(\)](#).

The ADM host application may supply a suite of AGM functions, including AGM image support functions. If not, this function cannot be used.

Parameters

inImage	An ADM image.
outImageRecord	The structure that describes inImage . Type: _t_ADMAGMImageRecord (see ADMAGMTypes.h)

Returns

None.

See also

[sADMImage->EndBaseAddressAccess\(\)](#)
[sADMImage->GetPixel\(\)](#)

sADMImage->BeginBaseAddressAccess()

Get the starting address of an image in memory

```
ASBytePtr ASAPI (*BeginBaseAddressAccess)(ADMImageRef
inImage);
```

Description

The **BeginBaseAddressAccess()** function accesses the beginning address of **inImage** in memory. The image is locked in memory and can safely be accessed and altered using the pointer that this function returns. Be sure to call [sADMImage->EndBaseAddressAccess\(\)](#) when this access is no longer needed so that appropriate resources are released.

Parameters

inImage	An ADM image.
----------------	---------------

Returns

Pointer to the beginning address of **inImage** in memory. Type: **ASBytePtr** (see `ASTypes.h`)

See also

[sADMImage->EndBaseAddressAccess\(\)](#)

sADMImage->Create()

Create a new image

```
ADMImageRef ASAPI (*Create)(ASInt32 inWidth, ASInt32 inHeight,  
ASInt32 inOptions);
```

Description

The **Create()** function creates a new image using **inWidth** and **inHeight**. A reference to the new image is returned. The image always has a color depth of 8-bits.

Parameters

inWidth	Width of the new image (in pixels).
inHeight	Height of the new image (in pixels).
inOptions	Pass 0 (zero) or kADMImageHasAlphaChannelOption (1L << 0).

Returns

The newly created ADM image.

See also

[sADMImage->CreateBitmap\(\)](#)
[sADMImage->CreateOffscreen\(\)](#)
[sADMImage->Destroy\(\)](#)

sADMImage->CreateBitmap()

Create bitmap image of specified size

```
ADMImageRef ASAPI (*CreateBitmap)(ASInt32 inWidth, ASInt32
inHeight, ASInt32 inOptions);
```

Description

The **CreateBitmap()** function creates a bit map image (1-bit in depth) of the given size.

Parameters

inWidth	Width of the new image (in pixels).
inHeight	Height of the new image (in pixels).
inOptions	Currently unused. Always pass 0 (zero).

Returns

The newly created ADM image.

See also

[sADMImage->Create\(\)](#)
[sADMImage->CreateOffscreen\(\)](#)
[sADMImage->Destroy\(\)](#)

sADMImage->CreateOffscreen()Create image of specified size
with current screen color depth

```
ADMImageRef ASAPI (*CreateOffscreen)(ASInt32 inWidth, ASInt32
inHeight, ASInt32 inOptions);
```

Description

The **CreateOffscreen()** function creates an image of the given size with the color depth of the current screen settings.

Parameters

inWidth	Width of the new image (in pixels).
inHeight	Height of the new image (in pixels).
inOptions	Pass 0 (zero) or kADMImageHasAlphaChannelOption .

Returns

The newly created ADM image.

See also

[sADMImage->Create\(\)](#)
[sADMImage->CreateBitmap\(\)](#)
[sADMImage->Destroy\(\)](#)

sADMImage->Destroy()

Remove an image from memory

```
void ASAPI (*Destroy)(ADMImageRef inImage);
```

Description

The **Destroy()** function removes an existing image from memory.

Parameters

inImage	An ADM image.
----------------	---------------

Returns

None.

See also

[sADMImage->Create\(\)](#)
[sADMImage->CreateBitmap\(\)](#)
[sADMImage->CreateOffscreen\(\)](#)

sADMImage->EndAGMImageAccess()

Stop using AGM routines to draw an image

```
void ASAPI (*EndAGMImageAccess)(ADMImageRef inImage);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **EndAGMImageAccess()** function stops drawing **inImage** with AGM routines. The ADM host application may supply a suite of AGM functions, including AGM image support functions. If not, this function cannot be used.

Parameters

inImage	An ADM image.
----------------	---------------

Returns

None.

See also

[sADMImage->BeginAGMImageAccess\(\)](#)

sADMImage->EndBaseAddressAccess()

Find out the end address of an image in memory

```
void ASAPI (*EndBaseAddressAccess)(ADMImageRef inImage);
```

Description

The **EndBaseAddressAccess()** function stops using the a base address to access **inImage**. Access began with a call to [sADMImage->EndBaseAddressAccess\(\)](#).

Parameters

inImage	An ADM image.
----------------	---------------

Returns

None.

See also

[sADMImage->BeginBaseAddressAccess\(\)](#)

[sADMImage->GetPixel\(\)](#)

sADMImage->EndADMDrawer()

Stop using an ADM Drawer to draw an image

```
void ASAPI (*EndADMDrawer)(ADMImageRef inImage);
```

Description

The **EndADMDrawer()** function stops enabling use of the ADM Drawer suite to draw **inImage**.

Parameters

inImage	An ADM image.
----------------	---------------

Returns

None.

See also

[sADMImage->BeginADMDrawer\(\)](#)

sADMImage->GetBitsPerPixel()

Get the bits per pixel of an image

```
ASInt32 ASAPI (*GetBitsPerPixel)(ADMImageRef inImage);
```

Description

The **GetBitsPerPixel()** function gets the bit depth of **inImage**. For example, a 10-pixel by 10-pixel image may have 8-bits per pixel resolution (256 colors), 16-bits per pixel (16,384 colors), and so on.

Parameters

inImage	An ADM image.
----------------	---------------

Returns

Bit depth of **inImage**.

sADMImage->GetByteWidth()

Get the width in bytes of an image

```
ASInt32 ASAPI (*GetByteWidth)(ADMImageRef inImage);
```

Description

The **GetByteWidth()** function gets the number of bytes wide **inImage** is. For example, if a black and white image is 10 pixels by 10 pixels, then its byte width is 2 (two bytes are required to hold 10 bits).

Parameters

inImage	An ADM image.
----------------	---------------

Returns

The number of bytes wide **inImage** is. $((\text{Width in pixels})/8 + 1)$

sADMImage->GetHeight()

Get the height of an image

```
ASInt32 ASAPI (*GetHeight)(ADMImageRef inImage);
```

Description

The **GetHeight()** function gets the height of **inImage**.

Parameters

inImage	An ADM image.
----------------	---------------

Returns

Height of **inImage**.

See also

[sADMImage->GetWidth\(\)](#)

sADMImage->GetPixel()

Get pixel data at a given point

```
ASErr ASAPI (*GetPixel)(ADMImageRef inImage, const ASPoint*
inPoint, ASRGBColor* outColor);
```

Description

The **GetPixel()** function obtains **outColor** for the pixel at **inPoint**. If you are working with all the pixels of an image, you may want to use the [sADMImage->BeginBaseAddressAccess\(\)](#) and [sADMImage->EndBaseAddressAccess\(\)](#) routines for speed.

Parameters

inImage	An ADM image.
inPoint	The pixel for which to obtain color data. Type: ASPoint (see ASTypes.h)
outColor	The color of pixel at inPoint . Type: ASRGBColor (see ASTypes.h)

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred.

See also

[sADMImage->SetPixel\(\)](#)
[sADMImage->BeginBaseAddressAccess\(\)](#)
[sADMImage->EndBaseAddressAccess\(\)](#)

sADMImage->GetWidth()

Get the width of an image

```
ASInt32 ASAPI (*GetWidth)(ADMImageRef inImage)
```

The **GetWidth()** function gets the width of **inImage**.

Parameters

inImage	An ADM image.
----------------	---------------

Returns

Width of **inImage** (in pixels).

See also

[sADMImage->GetHeight\(\)](#)

sADMImage->SetPixel()

Set pixel data at a given point

```
ASErr ASAPI (*SetPixel)(ADMImageRef inImage, const ASPoint*
inPoint, const ASRGBColor* inColor);
```

Description

The **SetPixel()** function sets the color data (**inColor**) for the pixel at **inPoint**.

Parameters

inImage	An ADM image.
inPoint	The pixel for which to set color data. Type: ASPoint (see ASTypes.h)
inColor	The color for pixel at inPoint . Type: ASRGBColor (see ASTypes.h)

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred.

See also

[sADMImage->GetPixel\(\)](#)

14

The ADM Item Suite

About the ADM Item Suite

The ADM Item suite allows you to create and access ADM Item objects. Many of the functions are those common to all ADM objects, such as text access functions. Others are unique to dialog items—for instance, setting text item numerics and picture IDs. This function reference builds on ideas established in [Chapter 1, “ADM Overview”](#).

Accessing the Suite

The ADM Item suite is referred to as:

```
#define kADMItemSuite          "ADM Item Suite"
```

with the version constant:

```
#define kADMItemSuiteVersion2      2
```

NOTE: Determine the suite version number you are using by examining the `ADMItem.h` header file.

The suite is acquired as follows:

```
ADMItemSuite *sADMItem;  
error = SSPBasic->AcquireSuite(kADMItemSuite, kADMItemSuiteVersion2,  
    &sADMItem);  
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

Initializing An Item

When you create a modal or non-modal ADM dialog, ADM automatically creates ADM items according to the dialog's item list resource. These items will have the default initialization. This means minimal initialization, so the dialog initialization function that you provide needs to further initialize the items.

Exactly what you need to initialize depends on the item type. You may want to set an item style if it cannot be set from the resource. All items can be enabled or disabled. You may want to make one edit text item active. Also, any item that needs to interact with other items should have a notification function. Item characteristics you might want to initialize for a given item are listed below:

TABLE 14.1 Initialization of ADM Items

Item Type (see <code>ADMItem.h</code>)	Initialization
<code>kADMDialType</code>	Set the dial's minimum, maximum, and current value. Assign a notification proc.
<code>kADMFrameType</code>	Likely nothing to initialize. You can set the style in the resource. You may want to set the frame text if it has a title.
<code>kADMItemGroupType</code>	<p>Nothing needs to be done in an item group's <code>InitProc</code>. At the time the <code>Init</code> proc is called, no child items have been added, so there's no way to apply any properties to them as a group. Since previously assigned item group properties are not propagated to child items when they're added, there's no point in setting any of those values before all the children are added. Perform the following steps to create an item group:</p> <ol style="list-style-type: none"> 1. <code>sADMItem->Create()</code> the item group (this causes the <code>InitProc()</code> to be called prior to return from <code>sADMItem->Create()</code>) 2. Add any child items that need to be added to this group (use <code>sADMItem->AddItem()</code>) 3. Set any properties for the group that should be propagated to the children (using the various ADM Item suite Set mutators).
<code>kADMTabbedMenuType</code>	Obsolete.
<code>kADMListBoxType</code>	Get the item's list and set its menu ID. If necessary, activate an entry or entries. Assign a notification function for the entries.
<code>kADMHierarchyListBoxType</code>	Get the item's list and set its menu ID. If necessary, activate an entry or entries. Assign initialization, notification and drawer functions for the entries.
<code>kADMPicturePushButtonType</code>	Set the picture ID. You may want to disable and select picture IDs as well. Also, a push button needs a notification function.
<code>kADMPictureCheckBoxType</code>	Set the picture ID. You may want to disable and select picture IDs as well. It might or might not need a notification function.

TABLE 14.1 Initialization of ADM Items

Item Type (see <code>ADMItem.h</code>)	Initialization
<code>kADMPictureRadioButtonType</code>	Set the picture ID. You may want to disable and select picture IDs as well. Set the boolean value of one radio button in a group to be true . You don't need to use a notification routine to handle radio button groups if all buttons in the group have consecutive IDs; ADM handles this case automatically.
<code>kADMPictureStaticType</code>	Set the picture ID. You may want to disable and select picture IDs as well. You may want a notification for an easter egg.
<code>kADMPopupControlType</code>	Set the usual (enable/disable, set Notify proc). By default, this type of item has a popup slider, so you set the min and max values to specify the range of the whole item. See Set***Value() functions.
<code>kADMPopupControlButtonType</code>	Set the usual (enable/disable, set Notify proc). By default, this type of item has a popup slider, so you set the min and max values to specify the range of the whole item. See Set***Value() functions.
<code>kADMPopupSpinEditControlType</code>	Set the usual (enable/disable, set Notify proc). Set the item's text. Note that these types of items can have only numeric data (floats or ints, negative values are acceptable). Use sADMItem->SetText() or Set***Value() functions. By default, this type of item has a popup slider, so you set the min and max values to specify the range of the whole item.
<code>kADMPopupListType</code>	Get the item's list and set its menu ID. If necessary, activate an entry or entries.
<code>kADMPopupMenuType</code>	Get the item's list and set its menu ID. If necessary, activate an entry or entries.
<code>kADMScrollingPopupMenuType</code>	Get the item's list and set its menu ID. If necessary, activate an entry or entries.
<code>kADMResizeType</code>	Set a notification function for the item.

TABLE 14.1 Initialization of ADM Items

Item Type (see <code>ADMItem.h</code>)	Initialization
<code>kADMScrollbarType</code>	Set the item's range and value. Set the large and small increments. Assign notification and tracking functions. Set the item's draw function for graphic feedback (e.g., a current picture).
<code>kADMSliderType</code>	Set the item's range and value. Assign a notification function if necessary. Set the item's draw function if it has graphic feedback (e.g., a color slider).
<code>kADMSpinEditType</code>	Set the item's text. Set any numerics, including ranges, units, etc. Set the increments to be used for the spinner.
<code>kADMSpinEditPopupType</code>	Get the item's list and set its menu ID. If necessary, activate an entry or entries. Set the item's text. Set any numerics, including ranges, units, etc. Set the small increment to be used for the spinner.
<code>kADMSpinEditScrollingPopupType</code>	Get the item's list and set its menu ID. If necessary, activate an entry or entries. Set the item's text. Set any numerics, including ranges, units, etc. Set the small increment to be used for the spinner.
<code>kADMTextCheckBoxType</code>	Set the item's boolean value.
<code>kADMTextEditType</code>	Set the item's text and maximum text length. Set any numerics, including ranges, units, and precision.
<code>kADMTextEditReadOnlyType</code>	Same as <code>kADMTextEditType</code> except that you can't type in them and you can't select them.
<code>kADMTextEditMultiLineType</code>	Set the item's text and maximum text length. Set any numerics, including ranges, units, and precision.
<code>kADMTextEditMultiLineReadOnlyType</code>	Same as <code>kADMTextEditMultiLineType</code> except you can't type in them and you can't select them.
<code>kADMTextEditPopupType</code>	Get the item's list and set its menu ID. If necessary, activate an entry or entries. Set the item's text. Set any numerics, including ranges, units, and precision.

TABLE 14.1 Initialization of ADM Items

Item Type (see <code>ADMItem.h</code>)	Initialization
<code>kADMTextEditScrollingPopUpType</code>	Get the item's list and set its menu ID. If necessary, activate an entry or entries. Set the item's text. Set any numerics, including ranges, units, and precision.
<code>ADMTextPushButtonType</code>	Push buttons need a notification routine.
<code>ADMTextRadioButtonType</code>	Set the boolean value of one radio button in a group to be true . You don't need to use a notification routine to handle radio button groups if all buttons in the group have consecutive IDs; ADM handles this case automatically.
<code>ADMTextStaticType</code>	Possibly enable or disable.
<code>kADMTextStaticMultilineType</code>	Possibly enable or disable.
<code>kADMProgressBarType</code>	Set the initial value. This item type typically exists transiently while an operation is being performed.
<code>kADMChasingArrowsType</code>	Only implemented on the Mac. This item would typically be used transiently to indicate that the user needs to wait. No initialization required.
<code>ADMUserType</code>	Since this is usually used for custom items, you set the draw, notification, and tracking functions. User items have the same properties as any other items, so you may also want to set a picture, text, or value. You determine the needed initialization.
<code>kADMPasswordTextEditType</code>	Somewhat like <code>kADMTextEditType</code> , except that passwords are typically not numeric and don't have default values. Set the maximum length in addition to the usual (enable/disable, set Notify proc).

FloatToText and TextToFloat Functions

The ADM `ADMItemTextToFloatProc()`/`ADMItemFloatToTextProc()` (see `ADMItem.h`) routines are available to plug-in programmers who want to override the ADM default text to float and float to text routine behaviors. You can override them by using the `sADMItem->SetTextToFloatProc()` and `sADMItem->SetFloatToTextProc()` of the ADM Item suite. With `ADMItemFloatToTextProc()`, you can affect the float

value used for the item and, reciprocally, with `ADMItemTextToFloatProc()` you can affect the text that is made visible to the user.

For instance, these would be used when you have an ADM numeric text item that has a min and max, but can also have no value to indicate that it is unused. Under the default ADM text to float routine, when the user deletes the value, ADM puts a 0 back in the field. To keep it from doing this, use `ADMItemTextToFloatProc()`.

If the `ADMItemTextToFloatProc()` returns `false`, the text is assumed to be invalid and a notification is presented to the user. If `true` is returned, and the item is known (see `sADMItem->IsKnown()`), ADM checks it against the min and max values and acts accordingly. If `true` is returned and your `ADMItemTextToFloatProc()` has marked the value as unknown, the text is used as is and no notification to the user is made.

The float-to-text functions are the following:

`sADMItem->SetFloatToTextProc()`

`sADMItem->GetFloatToTextProc()`

`sADMItem->DefaultFloatToText()`

`sADMItem->SetTextToFloatProc()`

`sADMItem->GetTextToFloatProc()`

`sADMItem->DefaultTextToFloat()`

ADM Help Support

NOTE: The Help API is deprecated in ADM V2.8. This does NOT include tool tips.

ADM supports the ASHelp online help system if it is available. This allows ADM UIs to use WinHelp compatible files to provide assistance to the user. Each ADM object can have a help ID that is used to identify a location in the help file to be displayed when help is triggered.

It is likely that the host application provides a help file for it and all the plug-ins that ship with it. Individual plug-ins can also provide their own help files that are independent of this.

To specify an alternate help file, a property is created in the plug-in's **PiPL**. If this property does not exist, help for the plug-in is assumed to be in the main help file.

The property is:

```
#define kHelpFileStrIDProperty 'HlpS'
```

The data for the property is versioned with the current specification:

```
//current is version 0
typedef struct PIHelpFileDesc
{
    long fVersion;
    long fFileNameStrID;
} PIHelpFileDesc
```

The **fFileNameStrID** is an indexed string resource giving the name of the help file to use for the plug-in. The name should be the first string in the list:

```
#define kHelpNativeStrIndex          1
```

So, on Macintosh, in addition to the **PiPL**/property, a '**STR#**' resource is created with the indicated ID and the name of the help file as the first (and possibly only) string. On Windows, a string resource with the ID "fFileNameStrID + 1" identifies the help file. The plug-in help file must be in the same directory as the main application help file.

The ADM help support functions are the following:

```
sADMItem->SetHelpID()
sADMItem->GetHelpID()
sADMItem->Help()
sADMItem->SetTipString()
sADMItem->GetTipString()
sADMItem->GetTipStringLength()
sADMItem->EnableTip()
sADMItem->IsTipEnabled()
sADMItem->ShowToolTip()
sADMItem->HideToolTip()
```

ADM Item Suite Functions

sADMItem->AbortTimer()

Abort a timer

```
void ASAPI (*AbortTimer)(ADMItemRef inItem, ADMTimerRef
inTimer);
```

Description

The **AbortTimer()** function aborts a timer procedure. It is used if the event specified by the **inAbortMask** parameter in [sADMItem->CreateTimer\(\)](#) occurs or if you destroy your item before your timer expires.

Parameters

inItem	An ADM item.
inTimer	Timer ID associated with inItem . Type: ADMTimerRef (see ADMTypes.h)

Returns

None.

See also

[sADMItem->CreateTimer\(\)](#)

sADMItem->Activate()

Make an item active or inactive

```
void ASAPI (*Activate)(ADMItemRef inItem, ASBoolean
inActivate);
```

Description

The **Activate()** function activates **inItem**. Pass **true** to activate the item. There can only be one active item in a dialog. On Windows, this can be any item in the dialog. On Macintosh, only text edit items can be active. By activating one item, others are automatically inactivated. An active item receives notification when the user presses the **Enter** key.

Parameters

inItem	An ADM item.
inActivate	If true , inItem is activated; if false , it is inactivated.

Returns

None.

See also

[sADMItem->IsActive\(\)](#)

sADMItem->AddItem()

Add an item to a group

```
void ASAPI (*AddItem)(ADMItemRef inGroup, ADMItemRef inItem);
```

Description

The **AddItem()** function adds **inItem** to **inGroup**.

An ADM item group is ADM's way of collecting a number of items together that need to respond to calls as a group. For example, you might have five items that all need to be enable or disabled simultaneously. Once those items belong to a group, you just need to enable/disable the group.

This is not true of geometrical containment. Item groups really don't have any physical manifestation; they are simply a way of logically grouping items.

This routine is an example of how sometimes ADM's APIs are not very object-oriented. You can call this routine with *any* **ADMItemRef** as the first argument, but it can only do something sensible if the item ref provided is for an item of type **kADMItemGroupType**.

Parameters

inGroup	An ADM item group.
inItem	An ADM item.

Returns

None.

See also

[sADMItem->RemoveItem\(\)](#)

sADMItem->Create()

Create a new ADM item

```
ADMItemRef ASAPI (*Create)(ADMDialogRef inDialog, ASInt32
inItemID, ADMItemType inItemType, const ASRect* inBoundsRect,
ADMItemInitProc inInitProc, ADMUserData inUserData, ASInt32
inOptions);
```

Description

The **Create()** function creates a new ADM item in the specified **inDialog**. The **inItemID** argument is a reference ID of the object. If you are creating all the items in a dialog you can use numbers in sequence. If you are adding an item to an existing dialog, use the **kADMUniqueItemID** constant and ADM will generate an ID for you.

inItemType indicates the type of item to create. ADM provides built-in types as described in [Dialog Item Objects](#) in [Chapter 1, "ADM Overview"](#) and defined in **ADMItem.h**.

The boundary rectangle (**inBoundsRect**) is the position and size of the item within the containing dialog's coordinate space. The initialization procedure **inInitProc** is the initialization routine (if any) for the item. The user data **inUserData** is used to store custom information (if any) associated with this item.

The returned value is a reference to the created item. You can use this to further initialize the item with other ADM Item suite functions.

Parameters

inDialog	An ADM dialog.
inItemID	Reference ID of the object. When adding an item to an existing dialog, use the kADMUniqueItemID constant and ADM generates an ID for you.
inItemType	Item type. Type: ADMItemType (see ADMItem.h)
inBoundsRect	Position and size of the item within inDialog 's coordinate space.
inInitProc	A callback with the following signature (see ADMItem.h): ADMItemInitProc(ADMItemRef inItem); Use to further initialize or customize the item. If special initialization is not required, pass NULL . ADM always checks to ensure it has a non- NULL proc before calling the proc.
inUserData	A 4-byte value that ADM keeps with the item; you can pass a pointer to a block of memory or some other value and retrieve it later using the sADMItem->GetUserData() function. Type: ADMUserData (see ADMTypes.h)
inOptions	Pass 0 (zero), kADMPasswordEditCreateOption , which fixes a problem with kADMPasswordTextEdit controls that are created on the fly instead of in a resource, or kADMIgnoreRomanFontForcing (see sADMItem->IgnoreForceRoman()).

Returns

The newly created ADM item.

See also

[sADMItem->GetUserData\(\)](#)
[sADMItem->IgnoreForceRoman\(\)](#)

sADMItem->CreateTimer()

Create a timer

```
ADMTimerRef ASAPI (*CreateTimer)(ADMItemRef inItem, ASUInt32
inMilliseconds, ADMActionMask inAbortMask, ADMItemTimerProc
inTimerProc, ADMItemTimerAbortProc inTimerAbortProc, ASInt32
inOptions);
```


Description

The **CreateTimer()** function creates a timer for measuring time between events where time is kept in milliseconds, with a user supplied **inTimerProc** and **inTimerAbortProc**. If the delay succeeds (i.e., is not aborted) then the **inTimerProc** is executed. If the action specified by **inAbortMask** occurs, **inTimerAbortProc** is be called. The possible values for **inAbortMask** are the same as the tracker masks and are defined in **ADMTracker.h**.

Parameters

inItem	An ADM item.
inMilliseconds	Delay between events.
inAbortMask	Specifies actions that result in calling inTimerAbortProc . Type: ADMActionMask (see ADMTypes.h)
inTimerProc	Callback with the following signature (see ADMItem.h): ASBoolean ADMItemTimerProc(ADMItemRef inItem, ADMTimerRef inTimer); This callback is executed at the end of an inMilliseconds delay. Returns a boolean. If it returns true , then inTimerProc will be called again after inMilliseconds . If it returns false then inTimerProc will no longer be called.
inTimerAbortProc	Callback with the following signature (see ADMItem.h): ADMItemTimerAbortProc(ADMItemRef inItem, ADMTimerRef inTimer, ADMAction inAbortAction); This callback is executed if an inAbortMask action occurs before the inMilliseconds delay completes. The values for inAbortAction are of type ADMAction and are listed in ADMTracker.h
inOptions	Currently unused. Always pass 0 (zero).

Returns

None.

See also

[sADMItem->AbortTimer\(\)](#)

sADMItem->DefaultDraw()

Call ADM's default item draw function

```
void ASAPI (*DefaultDraw)(ADMItemRef inItem, ADMDrawerRef
inDrawer);
```

Description

The **DefaultDraw()** function calls **inItem**'s current default draw function from within your custom item drawing function. The arguments passed to the custom function are passed through to the **DefaultDraw()** call.

Most likely, you will call the default drawing routine within a custom drawing function to get the basic appearance of the item. Your draw function would then add to **inItem**'s appearance. The exception to this is if you are completely changing the appearance of an item, or if the item is a user item, in which case its default drawing function does nothing.

Parameters

inItem	An ADM item.
inDrawer	An ADM drawer.

Returns

None.

See also

[sADMItem->SetDrawProc\(\)](#)

Example

```
void doNothingDrawHandler(ADMItemRef inItem, ADMDrawerRef inDrawer) {
    sADMItem->DefaultDraw(inItem, inDrawer);
}
```

sADMItem->DefaultFloatToText()

ADM's default procedure for converting a floating point number to text

```
ASBoolean ASAPI (*DefaultFloatToText)(ADMItemRef inItem, float
inValue, char* outText, ASInt32 inMaxLength);
```

Description

The **DefaultFloatToText()** function is the default floating point-to-text conversion function normally called by ADM. See [FloatToText](#) and [TextToFloat Functions](#) for more information.

Parameters

inItem	An ADM item.
inValue	Floating point value for conversion.
outText	inValue converted to text.
inMaxLength	Size of outText buffer.

Returns

None.

See also

[sADMItem->SetFloatToTextProc\(\)](#)

sADMItem->DefaultTextToFloat()

ADM's default procedure for converting a text string to a floating point number

```
ASBoolean ASAPI (*DefaultTextToFloat)(ADMItemRef inItem, const
char* inText, float* outValue);
```

Description

The **DefaultTextToFloat()** function is the default text-to-floating point function normally called by ADM. See [FloatToText](#) and [TextToFloat Functions](#) for more information.

Parameters

inItem	An ADM item.
inText	Text for conversion.
outValue	inText converted to a floating point number.

Returns

None.

See also

[sADMItem->SetTextToFloatProc\(\)](#)

sADMItem->DefaultNotify()

Call an ADM item's default notification function

```
void ASAPI (*DefaultNotify)(ADMItemRef inItem, ADMNotifierRef
inNotifier);
```

Description

The **DefaultNotify()** notify function calls **inItem**'s default notification function. Use this within a custom notification callback function. The arguments passed to the custom function are passed through to the **DefaultNotify()** call.

You will always call the default notification function for an item to get standard behaviors, such as list selection and radio button group behavior. You can skip calling **DefaultNotify()** only if you are creating your own item based on a user item or completely modifying the behavior of an existing item.

Parameters

inItem	AN ADM item.
inNotifier	An ADM notifier.

Returns

None.

See also

[sADMItem->SetNotifyProc\(\)](#)

Example

```
void doNothingNotificationHandler(ADMItemRef inItem,
    ADMNotifierRef inNotifier) {
    sADMItem->DefaultNotify(inItem, inNotifier);
    // Custom behavior goes here...
}
```

sADMItem->DefaultTrack()

Call the default tracker function for the ADM item

```
ASBoolean ASAPI (*DefaultTrack)(ADMItemRef inItem,
    ADMTrackerRef inTracker);
```

Description

The **DefaultTrack()** function calls the default tracking function of **inItem**. Use this within a custom tracker callback function. The arguments passed to the custom function are passed through to the **DefaultTrack()** call.

The behavior of a default tracker function depends on the item type. For instance, an ADM slider item's default tracker checks where the mouse-down event occurs and moves the thumb appropriately. An ADM button tracker inverts the button while the mouse is within the button and then returns it to normal when done.

An example of using a tracker function would be to have your function determine whether a modifier key is down, save the modifier state, and then call the default tracker. The default tracker would handle whether the item is to be notified by

returning **true** or **false**. When called, your item notifier function would check the saved modifier state and act upon it.

You don't have to call the default tracker function from within your custom tracker function; however, if you don't, be sure to call [sADMTracker->Abort\(\)](#). The **Abort()** function indicates that the end condition has been reached and causes the tracking loop to be exited.

Parameters

inItem	An ADM item.
inTracker	An ADM tracker.

Returns

Boolean result of the call.

See also

[sADMItem->SetTrackProc\(\)](#)

Example

```
ASBoolean doNothingTrackHandler(ADMItemRef inItem, ADMTrackerRef
inTracker) {
    bool rc = sADMItem->DefaultTrack(inItem, inTracker);

    if rc {
        //the default tracker is successfully called
    }
    else
    {
        //problem with execution of default tracker
    }

    return rc;
}
```

sADMItem->Destroy()

Remove an ADM item from memory

```
void ASAPI (*Destroy)(ADMItemRef inItem);
```

Description

The **Destroy()** function removes an ADM item from a dialog. If you have used [sADMItem->SetDestroyProc\(\)](#) to give the item a custom destroy callback, your function will be triggered by this call.

ADM automatically destroys all items in a dialog when the ADM notification for window closure is received. You would use this function only if you are creating and disposing of items dynamically in response to user actions.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

None.

See also

[sADMItem->SetDestroyProc\(\)](#)

sADMItem->Enable()

Enable or disable an item

```
void ASAPI (*Enable)(ADMItemRef inItem, ASBoolean inEnable);
```

Description

The **Enable()** function enables or disables **inItem**. Pass **true** to enable the item and **false** to disable it.

An enabled item can be selected by the user. A disabled item is dimmed and is unusable.

Parameters

inItem	An ADM item.
inEnable	If true , inItem is enabled; if false , inItem is disabled.

Returns

None.

See also

[sADMItem->IsEnabled\(\)](#)

sADMItem->GetAllowMath()

Gets whether an item uses math in computing numeric values

```
ASBoolean ASAPI (*GetAllowMath)(ADMItemRef inItem);
```

Description

The **GetAllowMath()** function determines whether math can be used in computing user input values for an item.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if math can be used in computing user input values for an item; **false** otherwise.

See also

[sADMItem->SetAllowMath\(\)](#)

sADMItem->GetAllowUnits()

Gets whether an item uses units in numeric values when values are entered by user

```
ASBoolean ASAPI (*GetAllowUnits)(ADMItemRef inItem);
```

Description

The **GetAllowUnits()** function gets whether units will be used for an item when the value is entered by the user.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if units are used for an item when the value is entered by the user; **false** otherwise.

See also

[sADMItem->SetAllowUnits\(\)](#)

sADMItem->GetBackColor()

Gets an item's background color

```
ADMColor ASAPI (*GetBackColor)(ADMItemRef inItem);
```

Description

The **GetBackColor()** function gets an item's background color.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

inItem's background color. Type: **ADMColor** (see **ADMTypes.h**)

See also

[sADMItem->SetBackColor\(\)](#)

sADMItem->GetBestSize()

Get the best size for an item

```
void ASAPI (*GetBestSize)(ADMItemRef inItem, ASPoint*
outBestSize);
```

Description

The **GetBestSize()** function gets the optimal dimensions for **inItem**.

For most items, this value is generated considering the text of the item and its graphic elements' bounds (e.g., a button's frame or a check box). For picture items, it returns the size of the largest associated picture.

Parameters

inItem	An ADM item.
outBestSize	Optimal dimensions for inItem . Type: ASPoint (see ASTypes.h)

Returns

None.

sADMItem->GetBooleanValue()

Get the boolean value of an item

```
ASBoolean ASAPI (*GetBooleanValue)(ADMItemRef inItem);
```

Description

The **GetBooleanValue()** function gets an item's boolean state, returning **true** (1) or **false** (0). This function is typically used with two state ADM items (such as check boxes and radio buttons). If used with other items, it returns **true** if the item's value is any non-zero number.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true or **false** to indicate the state of the item.

See also[sADMItem->SetBooleanValue\(\)](#)

sADMItem->GetBoundsRect()

Get the absolute position and size of an item

```
void ASAPI (*GetBoundsRect)(ADMItemRef inItem, ASRect*  
outBoundsRect);
```

Description

The **GetBoundsRect()** function gets the current size and position of an item in its containing dialog's coordinate space.

Parameters

inItem	An ADM item.
outBoundsRect	The current size and position of an item in its containing dialog's coordinate space. Type: ASRect (see ASTypes.h)

Returns

None.

See also[sADMItem->SetBoundsRect\(\)](#)

sADMItem->GetChildItem()

Get a child item of an item

```
ADMItemRef ASAPI (*GetChildItem)(ADMItemRef inItem, ASInt32  
inChildID);
```

Description

The **GetChildItem()** function returns a child item of a composite item based on the **inChildID** argument passed to it. Composite ADM items have a number of associated items called child items.

The ADM items with children include:

```
ADMListBox;  
ADMSpinEdit;  
ADMSpinEditPopup;  
ADMTextEditPopup;
```

Child item IDs are located in the file **ADMItem.h**. For instance, the IDs for an **ADMSpinEdit** item are:

```
typedef enum
{
    kADMSpinEditUpButtonChildID          = 1,
    kADMSpinEditDownButtonChildID        = 2,
    kADMSpinEditTextEditChildID          = 3,
    kADMSpinEditDummyChildID             = 0xFFFFFFFF
}
ADMSpinEditChildID;
```

The child items of a **ListBox**, a **SpinEditPopup** and **TextEditPopup** item include:

```
typedef enum
{
    kADMListBoxScrollbarChildID           = 1,
    kADMListBoxListChildID                = 2,
    kADMListBoxDummyChildID               = 0xFFFFFFFF
}
ADMListBoxChildID;
```

```
typedef enum
{
    kADMSpinEditPopupUpButtonChildID = 1,
    kADMSpinEditPopupDownButtonChildID= 2,
    kADMSpinEditPopupTextEditChildID= 3,
    kADMSpinEditPopupPopupChildID    = 4,
    kADMSpinEditPopupDummyChildID     = 0xFFFFFFFF
}
ADMSpinEditPopupChildID;
```

```
typedef enum
{
    kADMTextEditPopupTextEditChildID= 3,
    kADMTextEditPopupPopupChildID    = 4,
    kADMTextEditPopupDummyChildID     = 0xFFFFFFFF
}
```

Once a reference to a child item is obtained, any of the ADM Item suite functions can be used to access it.

Parameters

inItem	An ADM item.
inChildID	ID of child of composite item to return.

Returns

The child ADM item.

sADMItem->GetCursorID()

Get the item's cursor ID

```
void ASAPI (*GetCursorID)(ADMItemRef inItem, SPPluginRef*  
outPluginRef, ASInt32* outCursorID, const char**  
outCursorName);
```

Description

The **GetCursorID()** function gets the resource ID of the cursor to be displayed when the mouse position is inside the item.

Parameters

inItem	An ADM item.
outPluginRef	Plug-in reference.
outCursorID	Resource ID of the cursor to be displayed when the mouse position is inside the item.
outCursorName	Not currently used.

Returns

None.

See also

[sADMItem->SetCursorID\(\)](#)

sADMItem->GetDestroyProc()Get the ADM destroy function being
used for the item

```
ADMItemDestroyProc ASAPI (*GetDestroyProc)(ADMItemRef inItem);
```

Description

The **GetDestroyProc()** function returns the destroy function being used for **inItem**.

Because ADM calls the item's destroy function when it is disposed, you should not call the returned function directly.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The Destroy proc for **inItem**. If you have not called [sADMItem->SetDestroyProc\(\)](#), returns **NULL** (not the default Destroy proc). Type: **ADMItemDestroyProc** (see `ADMItem.h`)

See also

[sADMItem->SetDestroyProc\(\)](#)

sADMItem->GetDialog()

Get the dialog of an item

```
ADMDialogRef ASAPI (*GetDialog)(ADMItemRef inItem);
```

Description

The **GetDialog()** function gets the reference for a dialog containing the indicated item. The returned **ADMDialogRef** can be used with the ADM Dialog suite functions (see [Chapter 7, “The ADM Dialog Suite”](#)).

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The ADM dialog containing **inItem**.

sADMItem->GetDisabledPicture()

Set the item's disabled picture

```
ADMIconRef ASAPI (*GetDisabledPicture)(ADMItemRef inItem);
```

Description

The **GetDisabledPicture()** function gets the picture to be displayed for **inItem** when it is disabled.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

ADM icon that is displayed when **inItem** is disabled.

See also

[sADMItem->SetDisabledPicture\(\)](#)
[sADMItem->GetDisabledPictureID\(\)](#)
[sADMItem->GetPictureID\(\)](#)

```
sADMItem->GetPicture()  
sADMItem->GetSelectedPictureID()  
sADMItem->GetSelectedPicture()
```

sADMItem->GetDisabledPictureID()

Get the item's disabled picture ID

```
void WINAPI (*GetDisabledPictureID)(ADMItemRef inItem, ASInt32*  
outPictureResID, const char** outPictureResName);
```

Description

The **GetDisabledPictureID()** function gets the resource ID of the picture used to draw an item when it is disabled. If the item does not have a disabled picture, it returns 0.

Parameters

inItem	An ADM item.
outPictureResID	Resource ID for the picture. If the item does not have a disabled picture, it returns 0.
outPictureResName	Picture resource name.

Returns

Resource ID for the picture.

See also

```
sADMItem->SetDisabledPictureID()  
sADMItem->GetDisabledPicture()  
sADMItem->GetPictureID()  
sADMItem->GetPicture()  
sADMItem->GetSelectedPictureID()  
sADMItem->GetSelectedPicture()
```

sADMItem->GetDrawProc()Get the ADM drawing function being
used for the item

```
ADMItemDrawProc WINAPI (*GetDrawProc)(ADMItemRef inItem);
```

Description

The **GetDrawProc()** function gets the drawing function being used for **inItem**. Rather than getting and calling an item's drawing function in this fashion, you are more likely to use the **sADMItem->DefaultDraw()** function.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The Drawer proc for **inItem**. If you have not called [sADMItem->SetDrawProc\(\)](#), returns **NULL** (not the default Drawer proc). Type: **ADMItemDrawProc** (see `ADMItem.h`)

See also

[sADMItem->SetDrawProc\(\)](#)

[sADMItem->DefaultDraw\(\)](#)

sADMItem->GetFixedValue()

Get the fixed value of an item

```
ASFixed ASAPI (*GetFixedValue)(ADMItemRef inItem);
```

Description

The **GetFixedValue()** function returns the fixed value of **inItem**. If the item has an integer value, it is converted to a fixed decimal value.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The fixed value of **inItem**. Type: **ASFixed** (see `ASTypes.h`)

See also

[sADMItem->SetFixedValue\(\)](#)

sADMItem->GetFloatToTextProc()

Gets an item's procedure for converting a floating point value to a text string

```
ADMItemFloatToTextProc ASAPI (*GetFloatToTextProc)(ADMItemRef inItem);
```

Description

The **GetFloatToTextProc()** function gets the floating point to text conversion procedure for **inItem**. See [FloatToText and TextToFloat Functions](#) for more information.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Floating point to text conversion procedure for **inItem**. Type: **ADMItemFloatToTextProc** (see **ADMItem.h**)

See also

[sADMItem->SetFloatToTextProc\(\)](#)

sADMItem->GetFloatValue()

Get the float value of an item

```
float ASAPI (*GetFloatValue)(ADMItemRef inItem);
```

Description

The **GetFloatValue()** function returns the floating point value of **inItem**. If the item has an integer value, it is converted to a floating point decimal value.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The floating point value of **inItem**.

See also

[sADMItem->SetFloatValue\(\)](#)

sADMItem->GetFont()

Get the item's font style

```
ADMFont ASAPI (*GetFont)(ADMItemRef inItem);
```

Description

The **GetFont()** function returns **inItem**'s font style.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

inItem's font style. Type: **ADMFont** (see **ADMTypes.h**)

See also

[sADMItem->SetFont\(\)](#)

sADMItem->GetForeColor()

Get the item's foreground color

```
ADMCOLOR ASAPI (*GetForeColor)(ADMItemRef inItem);
```

Description

The **GetForeColor()** function returns **inItem**'s foreground color.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The color of the foreground. Type: **ADMCOLOR** (see **ADMTYPES.h**)

See also

[sADMItem->SetForeColor\(\)](#)

sADMItem->GetHasRolloverProperty()

Get whether the item has the rollover property

```
ASBoolean ASAPI (*GetHasRollOverProperty)(ADMItemRef inItem);
```

Description

The **GetHasRollOverProperty()** gets whether **inItem** has the rollover property. A control is in a rollover state when the mouse hovers over it.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if **inItem** has the rollover property; **false** otherwise.

See also

[sADMItem->SetHasRolloverProperty\(\)](#)
[sADMItem->GetRolloverPicture\(\)](#)
[sADMItem->GetRolloverPictureID\(\)](#)
[sADMItem->IsInRolloverState\(\)](#)

sADMItem->GetHierarchyList()

Get the hierarchy list for an item

```
ADMHierarchyListRef ASAPI (*GetHierarchyList)(ADMItemRef  
inItem);
```

Description

The **GetHierarchyList()** function returns the hierarchy list for **inItem**. With the returned reference, you can use the ADM Hierarchy List suite API with the item (see [Chapter 11, “The ADM Hierarchy List Suite”](#)).

NOTE: Not all items have hierarchy lists. This routine returns a non-**NULL** reference only if there's actually a hierarchy list associated with **inItem**.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

An ADM hierarchy list.

sADMItem->GetID()

Get the ID of an item

```
ASInt32 ASAPI (*GetID)(ADMItemRef inItem);
```

Description

The **GetID()** function returns the ID of **inItem**. This is the ID of the resource used to create it.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The ID of the resource used to create **inItem**.

sADMItem->GetIntValue()

Get the integer value of an item

```
ASInt32 ASAPI (*GetIntValue)(ADMItemRef inItem);
```

Description

The **GetIntValue()** function returns the integer value of **inItem**. If the item has a decimal value, it is converted to an integer.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The integer value of **inItem**.

See also

[sADMItem->SetIntValue\(\)](#)

sADMItem->GetItemStyle()

Get the style of an item

```
ADMItemStyle ASAPI (*GetItemStyle)(ADMItemRef inItem);
```

Description

The function returns the current style of an ADM item as discussed in [Dialog Item Objects](#) in [Chapter 1, “ADM Overview”](#) and [sADMItem->SetItemStyle\(\)](#).

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The item style. Type: **ADMItemStyle** (see [ADMItem.h](#))

See also

[sADMItem->SetItemStyle\(\)](#)

sADMItem->GetItemType()

Get the type of an item

```
ADMItemType ASAPI (*GetItemType)(ADMItemRef inItem);
```

Description

The **GetItemType()** function returns the current type of **inItem** as discussed in [Dialog Item Objects](#) in [Chapter 1, “ADM Overview”](#) and [sADMItem->Create\(\)](#).

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The type of **inItem**. Type: **ADMItemType** (see [ADMItem.h](#))

See also

[sADMItem->SetItemType\(\)](#)
[sADMItem->Create\(\)](#)

sADMItem->GetJustify()

Get the item's text justification

```
ADMJustify ASAPI (*GetJustify)(ADMItemRef inItem);
```

Description

The **GetJustify()** function returns **inItem**'s text justification.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The text justification for **inItem**. Type: **ADMJustify** (see `ADMTypes.h`)

See also

[sADMItem->SetJustify\(\)](#)

sADMItem->GetLargeIncrement()

Get the large increment value of an item

```
float ASAPI (*GetLargeIncrement)(ADMItemRef inItem);
```

Description

This function returns the large increment value of the item. It is only used for scroll bar items.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The large increment value of **inItem**.

See also

[sADMItem->SetLargeIncrement\(\)](#)
[sADMItem->GetSmallIncrement\(\)](#)
[sADMItem->SetSmallIncrement\(\)](#)

sADMItem->GetList()

Get the list of an item

```
ADMListRef ASAPI (*GetList)(ADMItemRef inItem);
```

Description

The **GetList()** function returns a reference to **inItem**'s ADM List object.

Once obtained, the ADM List suite functions can be used to access the list.

The following item types have valid list objects:

```
#define kADMListBoxType           "ADM List Box Type"
#define kADMPopupListType        "ADM Popup List Type"
#define kADMPopupMenuType        "ADM Popup Menu Type"
#define kADMScrollingPopupListType "ADM Scrolling Popup List Type"
#define kADMSpinEditPopupType     "ADM Spin Edit Popup Type"
#define kADMSpinEditScrollingPopupType
                                "ADM Spin Edit Scrolling Popup Type"
#define kADMTextEditPopupType     "ADM Text Edit Popup Type"
#define kADMTextEditScrollingPopupType
                                "ADM Text Edit Scrolling Popup Type"
```

Also, a custom item based on a user item might have a list object.

```
#define kADMUserType             "ADM User Type"
```

Parameters

inItem	An ADM item.
---------------	--------------

Returns

inItem's ADM List object.

sADMItem->GetLocalRect()

Get the size of an item

```
void ASAPI (*GetLocalRect)(ADMItemRef inItem, ASRect*
outLocalRect);
```

Description

The **GetLocalRect()** function gets the size of the dialog item in (0,0)-based dimensions. The **bottom** and **right** members of the ASRect structure are the item's size.

Parameters

inItem	An ADM item.
---------------	--------------

outLocalRect	The size of the dialog item in (0,0)-based dimensions. Type: ASRect (see <code>ASTypes.h</code>)
---------------------	---

Returns

None.

See also

[sADMItem->SetLocalRect\(\)](#)

sADMItem->GetMask()

Get the ADM notification mask for an item

```
ADMActionMask ASAPI (*GetMask)(ADMItemRef inItem);
```

Description

The **GetMask()** function returns the mask used for controlling which events will be received by the tracker.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The mask. Type: **ADMActionMask** (see `ADMTypes.h`)

See also

[sADMItem->SetMask\(\)](#)

sADMItem->GetMaxFixedValue()

Get the maximum fixed value of an item

```
ASFixed ASAPI (*GetMaxFixedValue)(ADMItemRef inItem);
```

Description

The **GetMaxFixedValue()** function returns the maximum fixed value of **inItem**. If **inItem** has an integer value, it is converted to a fixed point decimal value.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The maximum fixed value of **inItem**. Type: **ASFixed** (see `ASTypes.h`)

See also

[sADMItem->SetMaxFixedValue\(\)](#)

sADMItem->GetMaxFloatValue()

Get the maximum float value of an item

```
float ASAPI (*GetMaxFloatValue)(ADMItemRef inItem);
```

Description

The **GetMaxFloatValue()** function returns the maximum float value of **inItem**. If the item has an integer value, it is converted to a floating point decimal value.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The maximum float value for **inItem**.

See also

[sADMItem->SetMaxFloatValue\(\)](#)

sADMItem->GetMaxIntValue()

Get the maximum integer value of an item

```
ASInt32 ASAPI (*GetMaxIntValue)(ADMItemRef inItem);
```

Description

The **GetMaxIntValue()** function returns the maximum integer value of **inItem**. If the item has a decimal value, it is converted to an integer value.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The maximum integer value of **inItem**.

See also

[sADMItem->SetMaxIntValue\(\)](#)

sADMItem->GetMaxTextLength()

Get the maximum length of an item's text

```
ASInt32 ASAPI (*GetMaxTextLength)(ADMItemRef inItem);
```

Description

The **GetMaxTextLength()** function returns the maximum number of characters that **inItem**'s text property can have.

This is primarily of use for edit text items. The user is prohibited from entering more characters than the length.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Maximum number of characters that **inItem**'s text property can have.

See also

[sADMItem->SetMaxTextLength\(\)](#)

sADMItem->GetMinFixedValue()

Get the minimum fixed value of an item

```
ASFixed ASAPI (*GetMinFixedValue)(ADMItemRef inItem);
```

Description

The **GetMinFixedValue()** function returns the minimum fixed value of **inItem**. If the item has an integer value, it is converted to a fixed point decimal value.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The lower threshold on the value of **inItem**. Type: **ASFixed** (see `ASTypes.h`)

See also

[sADMItem->SetMinFixedValue\(\)](#)

sADMItem->GetMinFloatValue()

Get the minimum float value of an item

```
float ASAPI (*GetMinFloatValue)(ADMItemRef inItem);
```

Description

The **GetMinFloatValue()** function returns the minimum float value of **inItem**. If the item has an integer value, is converted to a floating point decimal value.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Lower threshold on the value of **inItem**.

See also

[sADMItem->SetMinFloatValue\(\)](#)

sADMItem->GetMinIntValue()

Get the minimum integer value of an item

```
ASInt32 ASAPI (*GetMinIntValue)(ADMItemRef inItem);
```

Description

The **GetMinIntValue()** function returns the minimum integer value of **inItem**. If the item has a decimal value, it is converted to a integer.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Minimum integer value of **inItem**.

See also

[sADMItem->SetMinIntValue\(\)](#)

sADMItem->GetNotifierData()

Get notifier data of an item

```
ADMUserData ASAPI (*GetNotifierData)(ADMItemRef inItem);
```

Description

The **GetNotifierData()** gets the notifier data of **inItem**.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The notifier data. Type: **ADMUserData** (see `ADMTypes.h`)

See also[sADMItem->SetNotifierData\(\)](#)

sADMItem->GetNotifyProc()

Get the ADM notification function being used for the item

```
ADMItemNotifyProc WINAPI (*GetNotifyProc)(ADMItemRef inItem);
```

Description

The **GetNotifyProc()** function returns the notification function being used for **inItem**.

Rather than getting and calling an item's notification function directly, you are more likely to use the [sADMItem->DefaultNotify\(\)](#) function.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The notification function for **inItem**. If you have not called [sADMItem->SetNotifyProc\(\)](#), returns **NULL** (not the default Notify proc). Type: **ADMDialogNotifyProc** (see `ADMItem.h`)

See also[sADMItem->DefaultNotify\(\)](#)
[sADMItem->SetNotifyProc\(\)](#)

sADMItem->GetPicture()

Gets a picture based on an icon

```
ADMIconRef WINAPI (*GetPicture)(ADMItemRef inItem);
```

Description

The **GetPicture()** function returns an icon reference for a particular item's picture.

NOTE: Not all items have a picture, so this function returns **NULL** for any **inItem** without an associated picture.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

An ADM icon. Returns **NULL** for any **inItem** without an associated picture.

See also

[sADMItem->SetPicture\(\)](#)
[sADMItem->GetDisabledPicture\(\)](#)
[sADMItem->GetDisabledPictureID\(\)](#)
[sADMItem->GetPictureID\(\)](#)
[sADMItem->GetSelectedPicture\(\)](#)
[sADMItem->GetSelectedPictureID\(\)](#)

sADMItem->GetPictureID()

Get the item's picture ID

```
void ASAPI (*GetPictureID)(ADMItemRef inItem, ASInt32*
outPictureResID, const char** outPictureResName);
```

Description

The **GetPictureID()** function gets the resource ID of the picture used to draw an item. If the item does not use a picture, it returns 0.

Parameters

inItem	An ADM item.
outPictureResID	Resource ID for the picture. If the item does not have a disabled picture, it returns 0.
outPictureResName	Picture resource name.

Returns

None.

See also

[sADMItem->SetPictureID\(\)](#)
[sADMItem->GetDisabledPicture\(\)](#)
[sADMItem->GetDisabledPictureID\(\)](#)
[sADMItem->GetPicture\(\)](#)
[sADMItem->GetSelectedPicture\(\)](#)
[sADMItem->GetSelectedPictureID\(\)](#)

sADMItem->GetPluginRef()

Get the plug-in that created an item

```
SPPluginRef ASAPI (*GetPluginRef)(ADMItemRef inItem);
```

Description

The **GetPluginRef()** function returns the plug-in that added **inItem**. You might, for example, use the plug-in reference to send it a message.

See the *Adobe PICA Programmer's Guide and Reference* for more information on directly interfacing with a plug-in.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Plug-in reference.

See also

[sADMItem->SetPluginRef\(\)](#)

sADMItem->GetPopupDialog()

Get a popup dialog

```
ADMDialogRef ASAPI (*GetPopupDialog)(ADMItemRef inItem);
```

Description

The **GetPopupDialog()** gets the popup dialog associated with **inItem** (if the item has an associated popup dialog). Not all items have one.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

An ADM pop-up dialog.

See also

[sADMItem->SetPopupDialog\(\)](#)

sADMItem->GetPrecision()

Get the precision of an item

```
ASInt32 ASAPI (*GetPrecision)(ADMItemRef inItem);
```

Description

The **GetPrecision()** function gets the precision of **inItem**. The returned value is the number of digits that the user is allowed to enter following a decimal point. It only applies to numeric edit text items.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Number of digits that the user is allowed to enter following a decimal point.

See also

[sADMItem->SetPrecision\(\)](#)

sADMItem->GetRolloverPicture()

Get the rollover picture for an item

```
ADMIconRef ASAPI (*GetRolloverPicture)(ADMItemRef inItem);
```

Description

The **GetRolloverPicture()** gets the rollover picture for **inItem**. A control is in a rollover state when the mouse hovers over it.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

An ADM icon.

See also

[sADMItem->SetRolloverPicture\(\)](#)
[sADMItem->GetRolloverPictureID\(\)](#)
[sADMItem->GetHasRolloverProperty\(\)](#)
[sADMItem->IsInRolloverState\(\)](#)

sADMItem->GetRolloverPictureID()

Get the rollover picture ID of an item

```
void ASAPI (*GetRolloverPictureID)(ADMItemRef inItem, ASInt32* outPictureResID, const char** outPictureResName);
```

Description

The **GetRolloverPictureID()** gets the rollover picture ID for **inItem**. A control is in a rollover state when the mouse hovers over it.

Parameters

inItem	An ADM item.
outPictureResID	Resource ID for the picture.
outPictureResName	Picture resource name.

Returns

None.

See also

[sADMItem->SetRolloverPictureID\(\)](#)
[sADMItem->GetRolloverPicture\(\)](#)
[sADMItem->GetHasRolloverProperty\(\)](#)
[sADMItem->IsInRolloverState\(\)](#)

sADMItem->GetSelectedPicture() Gets the picture currently selected in the item

```
ADMIconRef ASAPI (*GetSelectedPicture)(ADMItemRef inItem);
```

Description

The **GetSelectedPicture()** function returns an icon reference for the picture that **inItem** will use when it is in the selected state. Not all items have associated pictures or a selected state, so this item returns **NULL** for many items.

See also the description for [sADMItem->GetSelectedPictureID\(\)](#).

Parameters

inItem	An ADM item.
---------------	--------------

Returns

An ADM icon.

See also

[sADMItem->SetSelectedPicture\(\)](#)
[sADMItem->GetPicture\(\)](#)
[sADMItem->GetPictureID\(\)](#)
[sADMItem->GetSelectedPictureID\(\)](#)
[sADMItem->GetDisabledPicture\(\)](#)
[sADMItem->GetDisabledPictureID\(\)](#)

sADMItem->GetSelectedPictureID()

Get the item's selected picture ID

```
void ASAPI (*GetSelectedPictureID)(ADMItemRef inItem, ASInt32*
outPictureResID, const char** outPictureResName);
```

Description

The **GetSelectedPictureID()** function gets the resource ID of the picture used to draw an item when it is selected. If the item does not have a selected picture, it returns 0.

Parameters

inItem	An ADM item.
outPictureResID	Resource ID for the picture. If the item does not have a picture, it returns 0.
outPictureResName	Picture resource name.

Returns

None.

See also

[sADMItem->SetSelectedPictureID\(\)](#)
[sADMItem->GetPicture\(\)](#)
[sADMItem->GetPictureID\(\)](#)
[sADMItem->GetSelectedPicture\(\)](#)
[sADMItem->GetDisabledPicture\(\)](#)
[sADMItem->GetDisabledPictureID\(\)](#)

sADMItem->GetSelectionRange()

Get the active text item's selection range

```
void ASAPI (*GetSelectionRange)(ADMItemRef inItem, ASInt32*
outSelStart, ASInt32* outSelEnd);
```

Description

The **GetSelectionRange()** function gets the start and end positions of an active edit text item's text selection. **outSelEnd** is equal to the last selected character and can be no more than the item's text length.

This API is for use with edit text items. The characters between the positions are selected. If there is no selection for the item, **outSelStart** is equal to **outSelEnd**, and they indicate the text insertion point.

Parameters

inItem	An ADM item.
outSelStart	Beginning of selection. NOTE: outSelStart precedes the first selected character and thus might be 0.
outSelEnd	End of selection. Equal to the last selected character and can be no more than the item's text length.

Returns

None.

See also

[sADMItem->SetSelectionRange\(\)](#)

sADMItem->GetShowUnits()

Get whether units are shown in the value fields of an item

```
ASBoolean ASAPI (*GetShowUnits)(ADMItemRef inItem);
```

Description

The **GetShowUnits()** function determines whether the value fields of **inItem** show the selected units for the field.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if the value fields of **inItem** show the selected units for the field; **false** otherwise.

See also

[sADMItem->ShowUnits\(\)](#)

sADMItem->GetSmallIncrement()

Get the small increment value of an item

```
float ASAPI (*GetSmallIncrement)(ADMItemRef inItem);
```

Description

The **GetSmallIncrement()** function returns the small increment value of **inItem**. It is only used for sliders, spinners, and scrollbars.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The small increment value of **inItem**.

See also

[sADMItem->SetSmallIncrement\(\)](#)
[sADMItem->GetLargeIncrement\(\)](#)
[sADMItem->SetLargeIncrement\(\)](#)

sADMItem->GetText()

Get the item's text

```
void ASAPI (*GetText)(ADMItemRef inItem, char* outText,
ASInt32 inMaxLength);
```

Description

The **GetText()** function retrieves **inItem**'s text into the already allocated buffer pointed to by **outText**. The size of the buffer is indicated by **inMaxLength**. **inItem** uses this text as described in [sADMItem->SetText\(\)](#).

Parameters

inItem	An ADM item.
outText	inItem 's text.
inMaxLength	Size of the already allocated buffer for outText .

Returns

None.

See also

[sADMItem->SetText\(\)](#)

sADMItem->GetTextLength()

Get the length of an item's text

```
ASInt32 ASAPI (*GetTextLength)(ADMItemRef inItem);
```

Description

The **GetTextLength()** function gets the number of characters in **inItem**'s text.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Number of characters in **inItem**'s text.

sADMItem->GetTextToFloatProc()

Gets an item's procedure for converting a text string to a floating point value

```
ADMItemTextToFloatProc ASAPI (*GetTextToFloatProc)(ADMItemRef  
inItem);
```

Description

The **GetTextToFloatProc()** function gets the text to floating point conversion procedure for a specified item. See [FloatToText](#) and [TextToFloat Functions](#) for more information.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Text to floating point conversion procedure for **inItem**. Type:
ADMItemTextToFloatProc (see `ADMItem.h`)

See also

[sADMItem->SetTextToFloatProc\(\)](#)

sADMItem->GetTrackProc()

Get the ADM tracker function being used for the item

```
ADMItemTrackProc ASAPI (*GetTrackProc)(ADMItemRef inItem);
```

Description

The **GetTrackProc()** function returns the event tracking function being used for an item. Rather than getting and calling an item's tracker function directly, you are more likely to use the [sADMItem->DefaultTrack\(\)](#) function.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The track proc for **inItem**. If you have not called [sADMItem->SetTrackProc\(\)](#), returns **NULL** (not the default track proc). Type: **ADMItemTrackProc** (see **ADMItem.h**)

See also

[sADMItem->SetTrackProc\(\)](#)

sADMItem->GetUnits()

Get the units of a text item

```
ADMUnits WINAPI (*GetUnits)(ADMItemRef inItem);
```

Description

The **GetUnits()** function gets the measurement units used for a text item.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The units of measurement for **inItem**. Type: **ADMUnits** (see **ADMTypes.h**)

See also

[sADMItem->SetUnits\(\)](#)

sADMItem->GetUserData()

Get the user data pointer for an item

```
ADMUserData WINAPI (*GetUserData)(ADMItemRef inItem);
```

Description

The **GetUserData()** function returns the 4-byte user value stored with **inItem**. It is initialized by the [sADMItem->Create\(\)](#) function.

The meaning of the value is defined by the item's creator. Most likely it is a pointer to a data structure—for instance, the plug-in's globals. For some items, it might be a simple 4-byte type, such as a long or a fixed number.

An ADM item's user data is independent of its dialog's data.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The 4-byte user value stored with **inItem**. Type: **ADMUserData** (see **ADMTypes.h**)

See also

[sADMItem->SetUserData\(\)](#)
[sADMItem->Create\(\)](#)

sADMItem->GetWantsFocus()

Get the indicator showing whether the item wants focus

```
ASBoolean ASAPI (*GetWantsFocus)(ADMItemRef inItem);
```

Description

The **GetWantsFocus()** function returns the indicator of whether **inItem** wants focus. This function name is a bit of a misnomer—it really means that, if set, the item *can* accept focus, not that it “desires” it.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true indicates that **inItem** wants focus; **false** indicates otherwise.

See also

[sADMItem->SetWantsFocus\(\)](#)

sADMItem->GetWindowRef()

Get the window of a item

```
ASWindowRef ASAPI (*GetWindowRef)(ADMItemRef inItem);
```

Description

The **GetWindowRef()** function returns the platform window reference for **inItem**. Use this to, for example, draw directly into the window.

On Macintosh, this is the window’s GrafPort:

```
typedef struct GrafPort *ASWindowRef;
```

On Windows, this is the same as a Windows HWND:

```
typedef void * ASWindowRef;
```

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Platform window reference for **inItem**. Type: **ASWindowRef** (see **ASTypes.h**)

See also

[sADMDialog->GetWindowRef\(\)](#)
[sADMDialog->SetWindowRef\(\)](#)

sADMItem->IgnoreForceRoman()

Override Roman script forcing of item fonts

```
void ASAPI (*IgnoreForceRoman)(ADMItemRef inItem, ASBoolean
inIgnoreForceRoman);
```

Description

The **IgnoreForceRoman()** function provides a mutator for overriding Roman script forcing of item fonts. Normally one would set this with the **kADMIgnoreRomanFontForcing** flag (see `ADMTypes.h`) set as a mask on the options argument in the item's [sADMItem->Create\(\)](#) call. Items that are created in dialogs that are defined by platform resources do not have that ability, however—thus the mutator. Roman font forcing may be set globally by the application on a per-dialog basis. You may have a need to override on some items, so do that either via [sADMItem->Create\(\)](#) or do it after the fact with this call.

Parameters

inItem	An ADM item.
inIgnoreForceRoman	If true , Roman script forcing of fonts is overridden; if false , it is not.

Returns

None.

See also

[sADMItem->Create\(\)](#)
[sADMDrawer->Create\(\)](#)

sADMItem->Invalidate()

Invalidate the area of an item

```
void ASAPI (*Invalidate)(ADMItemRef inItem);
```

Description

The **Invalidate()** function invalidates **inItem**'s bounds within the dialog's window. This causes it to be redrawn the next time the screen is updated.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

None.

See also

[sADMItem->InvalidateRect\(\)](#)
[sADMDialog->Invalidate\(\)](#)
[sADMItem->Update\(\)](#)

sADMItem->InvalidateRect()

Invalidate the area of an item

```
void ASAPI (*InvalidateRect)(ADMItemRef inItem, const ASRect*  
inInvalRect);
```

Description

The **InvalidateRect()** function invalidates **inInvalRect** within **inItem**. This causes that area to be redrawn the next time the screen is updated.

Parameters

inItem	An ADM item.
inInvalRect	A rectangle within inItem . Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMItem->Invalidate\(\)](#)
[sADMDialog->InvalidateRect\(\)](#)

sADMItem->IsActive()

Get whether or not an item is active

```
ASBoolean ASAPI (*IsActive)(ADMItemRef inItem);
```

Description

The **IsActive()** function determines whether **inItem** is currently active. If so it returns **true**; if not, it returns **false**.

To change its state, use [sADMItem->Activate\(\)](#) function.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if **inItem** is active; **false** otherwise.

See also

[sADMItem->Activate\(\)](#)

sADMItem->IsEnabled()

Get whether or not an item is enabled

```
ASBoolean ASAPI (*IsEnabled)(ADMItemRef inItem);
```

Description

The **IsEnabled()** function determines whether **inItem** is currently enabled. If so, it returns **true**; if not, it returns **false**.

To change its state, use the [sADMItem->Enable\(\)](#) function. A disabled ADM item is dimmed and unusable.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if **inItem** is enabled; **false** otherwise.

See also

[sADMItem->Enable\(\)](#)

sADMItem->IsInRolloverState()

Get whether the item is in the rollover state

```
ASBoolean ASAPI (*IsInRollOverState)(ADMItemRef inItem);
```

Description

The **IsInRollOverState()** gets whether **inItem** is currently rolled over.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if **inItem** is in the rollover state; **false** otherwise.

See also

[sADMItem->SetInRolloverState\(\)](#)
[sADMItem->GetRolloverPicture\(\)](#)
[sADMItem->GetRolloverPictureID\(\)](#)
[sADMItem->GetHasRolloverProperty\(\)](#)

sADMItem->IsKnown()

Get whether or not an item is known

```
ASBoolean ASAPI (*IsKnown)(ADMItemRef inItem);
```

Description

The **IsKnown()** function determines whether **inItem** is known. If so, it returns **true**; if not, it returns **false**.

An item is in a “known” state if it has a “good” or valid value. For example, setting a checkbox item to **known(checkboxItem, false)** sets it to an intermediate state. The checkbox item then becomes “known” when it is checked by the user. The only way for an item to become “unknown” is by using the **known(someItem, false)** API. As another example, if you set a text item to unknown, it clears itself.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if **inItem** is known; **false** otherwise.

See also

[sADMItem->Known\(\)](#)

sADMItem->IsVisible()

Get whether or not an item is visible

```
ASBoolean ASAPI (*IsVisible)(ADMItemRef inItem);
```

Description

The **IsVisible()** function determines whether **inItem** is currently visible. If so, it returns **true**; if not, it returns **false**.

To change its state, use the **Show()** function.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if **inItem** is visible; **false** otherwise.

See also

[sADMItem->Show\(\)](#)

sADMItem->Known()

Make an item known

```
void ASAPI (*Known)(ADMItemRef inItem, ASBoolean inKnown);
```

Description

The **Known()** function makes **inItem** known.

An item is in a “known” state if it has a “good” or valid value. For example, setting a checkbox item to **known(checkboxItem, false)** sets it to an intermediate state. The checkbox item then becomes “known” when it is checked by the user. The only way for an item to become “unknown” is by using the **known(someItem, false)** API. As another example, if you set a text item to unknown, it clears itself.

NOTE: In a **kADMSpinEditType** item, if you call **known(spinEditItem, false)**, ADM disables the spin buttons, because the value of the edit field cannot be incremented/decremented.

Parameters

inItem	An ADM item.
inKnown	If true , inItem is made known; if false , inItem is made unknown.

Returns

None.

See also

[sADMItem->IsKnown\(\)](#)

sADMItem->LocalToScreenPoint()

Convert an item point to screen coordinates

```
void ASAPI (*LocalToScreenPoint)(ADMItemRef inItem, ASPoint* ioPoint);
```

Description

The **LocalToScreenPoint()** function converts a point in **inItem** to a point in screen coordinates.

Parameters

inItem	An ADM item.
ioPoint	The point. Type: ASPoint (see ASTypes.h)

Returns

None.

See also

[sADMItem->ScreenToLocalPoint\(\)](#)
[sADMDialog->LocalToScreenPoint\(\)](#)

sADMItem->LocalToScreenRect()

Convert an item rectangle to screen coordinates

```
void ASAPI (*LocalToScreenRect)(ADMItemRef inItem, ASRect*  
ioRect);
```

Description

The **LocalToScreenRect ()** function converts a rectangle in **inItem**'s coordinates to a rectangle in screen coordinates.

Parameters

inItem	An ADM item.
ioRect	The rect. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMItem->ScreenToLocalRect\(\)](#)
[sADMDialog->LocalToScreenRect\(\)](#)

sADMItem->Move()

Set the absolute position of an item

```
void ASAPI (*Move)(ADMItemRef inItem, ASInt32 inHorizPosition,  
ASInt32 inVertPosition);
```

Description

The **Move ()** function changes **inItem**'s absolute position.

If the item is moved out of the dialog's bounds using this function, it will not be forced back onto the dialog.

Parameters

inItem	An ADM item.
inHorizPosition	New horizontal screen coordinate for inItem .
inVertPosition	New vertical screen coordinate for inItem .

Returns

None.

sADMItem->RemoveItem()

Remove an item from a group

```
void ASAPI (*RemoveItem)(ADMItemRef inGroup, ADMItemRef
inItem);
```

Description

The **RemoveItem()** removes **inItem** from **inGroup**.

An ADM item group is ADM's way of collecting a number of items together that need to respond to calls as a group. For example, you might have five items that all need to be enable or disabled simultaneously. Once those items belong to a group, you just need to enable/disable the group. This routine is an example of how sometimes ADM's APIs are not very object-oriented. You can call this routine with *any* **ADMItemRef** as the first argument, but it can only do something sensible if the item ref provided is for an item of type **kADMItemGroupType**.

Parameters

inGroup	An ADM item group.
inItem	An ADM item.

Returns

None.

See also

[sADMItem->AddItem\(\)](#)

sADMItem->ScreenToLocalPoint()

Convert a screen point to item coordinates

```
void ASAPI (*ScreenToLocalPoint)(ADMItemRef inItem, ASPoint*  
ioPoint);
```

Description

The **ScreenToLocalPoint()** function converts a point in screen coordinates to a point relative to **inItem**.

Parameters

inItem	An ADM item.
ioPoint	The point. Type: ASPoint (see ASTypes.h)

Returns

None.

See also

[sADMItem->LocalToScreenPoint\(\)](#)
[sADMDialog->ScreenToLocalPoint\(\)](#)

sADMItem->ScreenToLocalRect()

Convert a rectangle to item coordinates

```
void ASAPI (*ScreenToLocalRect)(ADMItemRef inItem, ASRect*  
ioRect);
```

Description

The **ScreenToLocalRect()** function converts a rectangle in screen coordinates to a rectangle in the coordinate space of **inItem**.

Parameters

inItem	An ADM item.
ioRect	The rect. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMItem->LocalToScreenRect\(\)](#)
[sADMDialog->ScreenToLocalRect\(\)](#)

sADMItem->SelectAll()

Select the text of an item

```
void ASAPI (*SelectAll)(ADMItemRef inItem);
```

Description

The **SelectAll()** function selects the characters of the active edit text **inItem**.

This is for use with edit text items. It highlights the characters in an edit text item. It only works if the edit text item is currently active.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

None.

See also

[sADMItem->Activate\(\)](#)

sADMItem->SendNotify()

Send a notification to an item

```
void ASAPI (*SendNotify)(ADMItemRef inItem, const char*
inNotifierType);
```

Description

The **SendNotify()** function sends a notification of the indicated **inNotifierType** to **inItem**.

The main notifier for ADM items is:

```
#define kADMUserChangedNotifier "ADM User Changed Notifier"
```

This and other ADM notifiers are defined in `ADMNotifier.h`; you can also define your own notifiers.

Parameters

inItem	An ADM item.
inNotifierType	The type of notification to send. (See <code>ADMNotifier.h</code> for the notification constants.)

Returns

None.

sADMItem->SetAllowMath()

Sets whether an item uses math in computing entered values

```
void ASAPI (*SetAllowMath)(ADMItemRef inItem, ASBoolean  
inAllowMath);
```

Description

The **SetAllowMath()** function sets whether math can be used when a user enters values for **inItem**. For example, when set, if the user enters 2+3 for an input value, ADM automatically converts the value to 5 and shows it in the associated text area of the item.

Parameters

inItem	An ADM item.
inAllow	true if math can be used in computing user input values for an item; false otherwise.

Returns

None.

See also[sADMItem->GetAllowMath\(\)](#)

sADMItem->SetAllowUnits()

Sets whether an item uses units in numeric values when value is entered by user

```
void ASAPI (*SetAllowUnits)(ADMItemRef inItem, ASBoolean  
inAllowUnits);
```

Description

The **SetAllowUnits()** function sets whether units are used for an item when the value is entered by the user.

Parameters

inItem	An ADM item.
inAllowUnits	true if units are used for an item when the value is entered by the user; false otherwise.

Returns

None.

See also

[sADMItem->GetAllowUnits\(\)](#)

sADMItem->SetBackColor()

Sets an item's background color

```
void ASAPI (*SetBackColor)(ADMItemRef inItem, ADMColor
inColor);
```

Description

The **SetBackColor()** function sets an item's background color.

Parameters

inItem	An ADM item.
inColor	inItem 's background color. Type: ADMColor (see ADMTTypes.h)

Returns

None.

See also

[sADMItem->GetBackColor\(\)](#)

sADMItem->SetBooleanValue()

Set the boolean value of an item

```
void ASAPI (*SetBooleanValue)(ADMItemRef inItem, ASBoolean
inValue);
```

Description

The **SetBooleanValue()** function sets an item's boolean state to **true** (1) or false (0). This function is used with two-state ADM items (such as check boxes and radio buttons).

Parameters

inItem	An ADM item.
inValue	true or false to indicate the state of the item.

Returns

None.

See also

[sADMItem->GetBooleanValue\(\)](#)

sADMItem->SetBoundsRect()

Set the absolute position and size of an item

```
void ASAPI (*SetBoundsRect)(ADMItemRef inItem, const ASRect*
inBoundsRect);
```

Description

The **SetBoundsRect()** function sets the size and position of the item relative to the containing dialog's bounds.

Because the size of the item is expressed in dialog coordinates, its dimensions must be computed:

```
width = inBoundsRect.right - inBoundsRect.left;
height = inBoundsRect.bottom - inBoundsRect.top;
```

To move the item in the dialog, you would first get the item's bounds rectangle and change it by the move amount.

```
sADMItem->GetBoundsRect(myItem, &inBoundsRect);
boundsRect.right += relativeMove.h;
boundsRect.left += relativeMove.h;
boundsRect.top += relativeMove.v;
boundsRect.bottom += relativeMove.v;
sADMItem->SetBoundsRect(myItem, &inBoundsRect);
```

It is simpler to do this using the [sADMItem->Move\(\)](#) function.

If the item is moved off the dialog using this function, it will not be forced back onto a visible part.

Parameters

inItem	An ADM item.
inBoundsRect	The size and position of an item in its containing dialog's coordinate space. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMItem->GetBoundsRect\(\)](#)
[sADMItem->Move\(\)](#)

sADMItem->SetCursorID()

Set the item's cursor ID

```
ASBoolean ASAPI (*SetCursorID)(ADMItemRef inItem, SPPluginRef
inPluginRef, ASInt32 inCursorID, const char* inCursorName);
```

Description

The **SetCursorID()** function sets the cursor to be displayed when the mouse position is inside the item. **inCursorID** is the ID of a platform native cursor resource. The actual platform resource itself is different for each platform, but you can call this API in the same way for all platforms.

NOTE: The object that sets the cursor ID must change it back to be the arrow cursor before some other object can set the cursor ID. When an object sets the cursor ID, it owns it until it resets it to the arrow cursor.

Parameters

inItem	An ADM item.
inPluginRef	Plug-in reference.
inCursorID	ID of a platform native cursor resource.
inCursorName	Cursor name. Can also be NULL .

Returns

true if the specified resource was found and the cursor was changed to use it;
false if there was any failure in the process.

See also

[sADMItem->GetCursorID\(\)](#)

sADMItem->SetDestroyProc()

Set the ADM destroy function to use for the item

```
void ASAPI (*SetDestroyProc)(ADMItemRef inItem,
ADMItemDestroyProc inDestroyProc);
```

Description

The **SetDestroyProc()** function assigns destroy function **inDestroyProc** to **inItem**. This callback is called when [sADMItem->Destroy\(\)](#) is invoked with the item's **ADMItemRef**. Use it to free memory and other resources you may have allocated for the item.

ADM will always destroy the item object, so you do not need to call a default destroy function.

Parameters

inItem	An ADM item.
inDestroyProc	A callback with the following signature (see <code>ADMItem.h</code>): ADMItemDestroyProc(ADMItemRef inItem);

Returns

None.

See also

[sADMItem->GetDestroyProc\(\)](#)

sADMItem->SetDisabledPicture()

Set the item's disabled picture

```
void WINAPI (*SetDisabledPicture)(ADMItemRef inItem, ADMIconRef
inPicture);
```

Description

The **SetDisabledPicture()** function sets the picture to be displayed for **inItem** when it is disabled. You provide the picture in the form of a previously created ADM icon.

NOTE: The same limited set of icons applies here as in [sADMItem->SetDisabledPictureID\(\)](#).

Parameters

inItem	An ADM item.
inPicture	ADM icon that is to be displayed when inItem is disabled.

Returns

None.

See also

[sADMItem->GetDisabledPicture\(\)](#)
[sADMItem->SetPictureID\(\)](#)
[sADMItem->SetPicture\(\)](#)
[sADMItem->SetSelectedPictureID\(\)](#)
[sADMItem->SetSelectedPicture\(\)](#)
[sADMItem->SetDisabledPictureID\(\)](#)

sADMItem->SetDisabledPictureID()

Set the item's disabled picture ID

```
void WINAPI (*SetDisabledPictureID)(ADMItemRef inItem, ASInt32
inPictureResID, const char* inPictureResName);
```

Description

The **SetDisabledPictureID()** function sets the picture to be displayed for the item when it is disabled. The **inPictureResID** is the ID of a platform picture or icon resource.

Not all ADM items have pictures. ADM items for which this function sets a picture resource are:

```
#define kADMPictureCheckBoxType    "ADM Picture Check Box Button Type"
#define kADMPicturePushButtonType  "ADM Picture Push Button Type"
#define kADMPictureRadioButtonType "ADM Picture Radio Button Type"
#define kADMPictureStaticType      "ADM Picture Static Type"
```

A custom item type is based on a user item. This may or may not use a picture:

```
#define kADMUserType                "ADM User Type"
```

If the item does not have a disabled picture, ADM grays the default picture when the item is disabled.

Parameters

inItem	An ADM item.
inPictureResID	Resource ID for the picture.
inPictureResName	Picture resource name.

Returns

None.

See also

```
sADMItem->GetDisabledPictureID()
sADMItem->SetDisabledPicture()
sADMItem->SetPictureID()
sADMItem->SetPicture()
sADMItem->SetSelectedPictureID()
sADMItem->SetSelectedPicture()
```

sADMItem->SetDrawProc()

Set the ADM drawing function to use for the item

```
void ASAPI (*SetDrawProc)(ADMItemRef inItem, ADMItemDrawProc
inDrawProc);
```

Description

The **SetDrawProc()** function defines drawing function **inDrawProc** for **inItem**.

Within your drawing function you can use the ADM Drawer suite APIs to perform standard image operations such as drawing lines and pictures. The **ADMDrawerRef**

argument is passed to the ADM Drawer suite functions to indicated where the imaging is to occur.

See [Using Event Callbacks](#) in [Chapter 1, “ADM Overview”](#) for more information.

Parameters

inItem	An ADM item.
inDrawProc	A callback with the following signature (see <code>ADMItem.h</code>): ADMItemDrawProc(ADMItemRef inItem, ADMDrawerRef inDrawer); The ADMDrawerRef argument is passed to ADM Drawer suite functions to indicated where the imaging is to occur.

Returns

None.

See also

[sADMItem->GetDrawProc\(\)](#)

sADMItem->SetFixedValue()

Set the fixed value of an item

```
void ASAPI (*SetFixedValue)(ADMItemRef inItem, ASFixed  
inValue);
```

Description

The **SetFixedValue()** function sets the value of the item to the supplied fixed number.

If a text field is set to a numeric value, the text representation of the value will be displayed. If it is a slider or scroll bar, the thumb will move to the correct position.

Parameters

inItem	An ADM item.
inValue	A fixed value for inItem . Type: ASFixed (see <code>ASTypes.h</code>)

Returns

None.

See also

[sADMItem->GetFixedValue\(\)](#)

sADMItem->SetFloatToTextProc()

Sets an item's procedure for converting a floating point value to a text string

```
void ASAPI (*SetFloatToTextProc)(ADMItemRef inItem,
ADMItemFloatToTextProc inProc);
```

Description

The **SetFloatToTextProc()** function sets the floating point to text conversion procedure for a specified item. See [FloatToText](#) and [TextToFloat Functions](#) for more information.

Parameters

inItem	An ADM item.
inProc	A callback with the following signature (see <code>ADMItem.h</code>): ASBoolean ADMItemFloatToTextProc(ADMItemRef inItem, float inValue, char* outText, ASInt32 inMaxLength); inMaxLength specifies the maximum length of outText .

Returns

None.

See also

[sADMItem->GetFloatToTextProc\(\)](#)
[sADMItem->DefaultFloatToText\(\)](#)

sADMItem->SetFloatValue()

Set the float value of an item

```
void ASAPI (*SetFloatValue)(ADMItemRef inItem, float inValue);
```

Description

The **SetFloatValue()** function sets the value of **inItem** to the supplied floating point number.

If a text field is set to a numeric value, the text representation of the value is displayed. If it is a slider or scroll bar, the thumb moves to the correct position.

Parameters

inItem	An ADM item.
inValue	The floating point value for inItem .

Returns

None.

See also

[sADMItem->GetFloatValue\(\)](#)

sADMItem->SetFont()

Set the item's font style

```
void ASAPI (*SetFont)(ADMItemRef inItem, ADMFont inFont);
```

Description

The **SetFont()** function sets the font used for **inItem**. Some items do not support this.

Parameters

inItem	An ADM item.
inFont	inItem 's font style. Type: ADMFont (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMItem->GetFont\(\)](#)

sADMItem->SetForeColor()

Set the item's foreground color

```
void ASAPI (*SetForeColor)(ADMItemRef inItem, ADMColor  
inColor);
```

Description

The **SetForeColor()** function sets **inItem**'s foreground color.

Parameters

inItem	An ADM item.
inColor	The color for the foreground. Type: ADMColor (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMItem->GetForeColor\(\)](#)

sADMItem->SetHasRolloverProperty()

Set whether the item has the rollover property

```
void ASAPI (*SetHasRollOverProperty)(ADMItemRef inItem,
ASBoolean inRollover);
```

Description

The **SetHasRollOverProperty()** sets whether **inItem** has the rollover property. A control is in a rollover state when the mouse hovers over it.

Parameters

inItem	An ADM item.
inRollover	If true , inItem has the rollover property; if false , it doesn't.

Returns

None.

See also

[sADMItem->GetHasRolloverProperty\(\)](#)
[sADMItem->SetRolloverPicture\(\)](#)
[sADMItem->SetRolloverPictureID\(\)](#)
[sADMItem->SetInRolloverState\(\)](#)

sADMItem->SetInRolloverState()

Set whether the item is in the rollover state

```
void ASAPI (*SetInRollOverState)(ADMItemRef inItem, ASBoolean
inRolloverState);
```

Description

The **SetInRollOverState()** sets whether **inItem** is currently rolled over.

Parameters

inItem	An ADM item.
inRolloverState	If true , inItem is in the rollover state; if false , it isn't.

Returns

None.

See also

[sADMItem->IsInRolloverState\(\)](#)
[sADMItem->SetRolloverPicture\(\)](#)
[sADMItem->SetRolloverPictureID\(\)](#)
[sADMItem->SetHasRolloverProperty\(\)](#)

sADMItem->SetIntValue()

Set the integer value of an item

```
void ASAPI (*SetIntValue)(ADMItemRef inItem, ASInt32 inValue);
```

Description

The **SetIntValue()** function sets the value of **inItem** to the supplied integer.

If a text field is set to a numeric value, the text representation of the value is displayed.
If it is a slider or scroll bar, the thumb moves to the correct position.

Parameters

inItem	An ADM item.
inValue	The integer value for inItem .

Returns

None.

See also

[sADMItem->GetIntValue\(\)](#)

sADMItem->SetItemStyle()

Set the style of an item

```
void ASAPI (*SetItemStyle)(ADMItemRef inItem, ADMItemStyle  
inItemStyle);
```

Description

The **SetItemStyle()** function sets the style of **inItem**. Within your dialog initialization function, call this function to set the item's style. Valid item styles depend on the item type and are described in [Dialog Item Objects](#) in [Chapter 1, "ADM Overview"](#) and defined in the `ADMItem.h` header file.

For example, an ADM Frame item has a number of styles:

```
#define kADMFrameType                "ADM Frame Type"
typedef enum
{
    kADMBlackFrameStyle              = 0,
    kADMGrayFrameStyle               = 1,
    kADMSunkenFrameStyle             = 2,
    kADMRaisedFrameStyle             = 3,
    kADMEtchedFrameStyle            = 4,
    kADMDummyFrameStyle              = 0xFFFFFFFF
}
ADMFrameStyle;
```

A plug-in adding a custom item would use this function to change the item style to its custom style. This function needs to be followed by [sADMItem->Update\(\)](#).

Parameters

inItem	An ADM item.
inItemStyle	The item style. Type: ADMItemStyle (see <code>ADMItem.h</code>)

Returns

None.

See also

[sADMItem->GetItemStyle\(\)](#)
[sADMItem->Update\(\)](#)

sADMItem->SetItemType()

Set the type of an item

```
void ASAPI (*SetItemType)(ADMItemRef inItem, ADMItemType
inItemType);
```

Description

The **SetItemType()** function sets the type **inItem**. Valid item types are given in [Dialog Item Objects](#) in [Chapter 1, "ADM Overview"](#) and `ADMItem.h`.

A custom item uses the **"ADM User Type"** item as a foundation:

```
#define kADMUserType "ADM User Type"
```


After an ADM item has been created, its type should not be changed using this routine *unless* its original item type is **kADMUserType**. Results are unpredictable if this routine is used to alter any other ADM item.

Parameters

inItem	An ADM item.
inItemType	The type for inItem . Type: ADMItemType (see ADMItem.h)

Returns

None.

See also

[sADMItem->GetItemType\(\)](#)
[sADMItem->Create\(\)](#)

sADMItem->SetJustify()

Set the item's text justification

```
void ASAPI (*SetJustify)(ADMItemRef inItem, ADMJustify
inJustify);
```

Description

The **SetJustify()** function sets **inItem**'s text justification.

Justification is useful for edit text and static text items. ADM justification constants are the following (see **ADMTypes.h**):

```
typedef enum
{
    kADMLeftJustify           = 0,
    kADMCenterJustify         = 1,
    kADMRightJustify          = 2,
    kADMDummyJustify          = 0xFFFFFFFF
}
ADMJustify;
```

NOTE: On Windows systems, this may not work due to a Microsoft operating system limitation.

Parameters

inItem	An ADM item.
inJustify	Text justification for inItem . Type: ADMJustify (see ADMTypes.h)

Returns

None.

See also

[sADMItem->GetJustify\(\)](#)

sADMItem->SetLargeIncrement()

Set the large increment value of an item

```
void WINAPI (*SetLargeIncrement)(ADMItemRef inItem, float
inIncrement);
```

Description

The **SetLargeIncrement()** function sets the large increment value of **inItem**. The large increment is used by scroll bar items and is the amount to add or subtract from the item value when the user clicks in the control area between the thumb and an arrow.

Parameters

inItem	An ADM item.
inIncrement	The large increment value for inItem .

Returns

None.

See also

[sADMItem->GetLargeIncrement\(\)](#)
[sADMItem->GetSmallIncrement\(\)](#)
[sADMItem->SetSmallIncrement\(\)](#)

sADMItem->SetLocalRect()

Set the size of an item

```
void WINAPI (*SetLocalRect)(ADMItemRef inItem, const ASRect*
inLocalRect);
```

Description

The **SetLocalRect()** function sets the size of **inItem**. Setting the size of the item based on the local rectangle means using a (0,0)-based rectangle of the absolute dimensions.

It is simpler to use the [sADMItem->Size\(\)](#) function to accomplish this.

Parameters

inItem	An ADM item.
inLocalRect	The size for the item in (0,0)-based dimensions. Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMItem->GetLocalRect\(\)](#)

sADMItem->SetMask()

Set the ADM notification mask for an item

```
void ASAPI (*SetMask)(ADMItemRef inItem, ADMActionMask  
inActionMask);
```

Description

The **SetMask()** function sets the mask for controlling which events will be received by the tracker. Maskable events are defined in **ADMTracker.h**.

Parameters

inItem	An ADM item.
inActionMask	The mask. Type: ADMActionMask (see ADMTypes.h)

Returns

None.

See also

[sADMItem->GetMask\(\)](#)

sADMItem->SetMaxFixedValue()

Set the maximum fixed value of an item

```
void ASAPI (*SetMaxFixedValue)(ADMItemRef inItem, ASFixed  
inValue);
```

Description

The **SetMaxFixedValue()** function sets an upper threshold on the value of **inItem** to the supplied fixed point value. If the user tries to set a value greater than this, it is automatically set to this value.

Parameters

inItem	An ADM item.
inValue	The maximum fixed value for inItem . Type: ASFixed (see ASTypes.h)

Returns

None.

See also

[sADMItem->GetMaxFixedValue\(\)](#)

sADMItem->SetMaxFloatValue()

Set the maximum float value of an item

```
void ASAPI (*SetMaxFloatValue)(ADMItemRef inItem, float
inValue);
```

Description

The **SetMaxFloatValue()** function sets an upper threshold on the value of **inItem** to the supplied floating point value. If the user tries to set a value greater than this, it is automatically set to this value.

Parameters

inItem	An ADM item.
inValue	The maximum float value for inItem .

Returns

None.

See also

[sADMItem->GetMaxFloatValue\(\)](#)

sADMItem->SetMaxIntValue()

Set the maximum integer value of an item

```
void ASAPI (*SetMaxIntValue)(ADMItemRef inItem, ASInt32
inValue);
```

Description

The **SetMaxIntValue()** function sets an upper threshold on the value of **inItem** to the supplied integer value. If the user tries to set a value greater than this, it is automatically set to this value.

Parameters

inItem	An ADM item.
inValue	The maximum integer value of inItem .

Returns

None.

See also

[sADMItem->GetMaxIntValue\(\)](#)

sADMItem->SetMaxTextLength()

Set the maximum length of an item's text

```
void ASAPI (*SetMaxTextLength)(ADMItemRef inItem, ASInt32  
inLength);
```

Description

The **SetMaxTextLength()** function sets the maximum number of characters that **inItem**'s text property can have.

This is primarily of use to edit text items. By setting the maximum text length of an item, the user is prohibited from entering more characters than the length.

Parameters

inItem	An ADM item.
inLength	Maximum number of characters that inItem 's text property can have.

Returns

None.

See also

[sADMItem->GetMaxTextLength\(\)](#)

sADMItem->SetMinFixedValue()

Set the minimum fixedvalue of an item

```
void ASAPI (*SetMinFixedValue)(ADMItemRef inItem, ASFixed  
inValue);
```

Description

The **SetMinFixedValue()** function sets the lower threshold on the value of **inItem** to the supplied fixed value. If the user tries to set a value below this, it is automatically set to this value.

Parameters

inItem	An ADM item.
inValue	The lower threshold on the value of inItem . Type: ASFixed (see ASTypes.h)

Returns

None.

See also

[sADMItem->GetMinFixedValue\(\)](#)

sADMItem->SetMinFloatValue()

Set the minimum float value of an item

```
void ASAPI (*SetMinFloatValue)(ADMItemRef inItem, float
inValue);
```

Description

The **SetMinFloatValue()** function sets a lower threshold on the value of **inItem** to the supplied float value. If the user tries to set a value below this, it is automatically set to this value.

Parameters

inItem	An ADM item.
inValue	Lower threshold on the value of inItem .

Returns

None.

See also

[sADMItem->GetMinFloatValue\(\)](#)

sADMItem->SetMinIntValue()

Set the minimum integer value of an item

```
void ASAPI (*SetMinIntValue)(ADMItemRef inItem, ASInt32
inValue);
```

Description

The **SetMinIntValue()** function sets a lower threshold on the value of **inItem** to the supplied integer value. If the user tries to set a value below this, it is automatically set to this value.

Parameters

inItem	An ADM item.
inValue	Minimum integer value of inItem .

Returns

None

See also

[sADMItem->GetMinIntValue\(\)](#)

sADMItem->SetNotifierData()

Set notifier data of an item

```
void ASAPI (*SetNotifierData)(ADMItemRef inItem, ADMUserData  
inData);
```

Description

The **SetNotifierData()** function sets the notifier data of **inItem**, if any. This is used for custom notification procedures.

Parameters

inItem	An ADM item.
inData	Custom notification data. Type: ADMUserData (see ADMTYPES.h)

Returns

None.

See also

[sADMItem->GetNotifierData\(\)](#)

sADMItem->SetNotifyProc()

Set the ADM notification function to use for the item

```
void ASAPI (*SetNotifyProc)(ADMItemRef inItem,  
ADMItemNotifyProc inNotifyProc);
```

Description

The `SetNotifyProc()` assigns event notification function **`inNotifyProc`** to the **`inItem`**. The notification is sent after a mouse-up event occurs on the item.

Within your notification function you can use the ADM Notifier suite functions to determine the type of notification received. The **`inNotifier`** argument is passed to the ADM Notifier suite functions to indicate the event for which information is being requested.

See [Using Event Callbacks](#) in [Chapter 1, “ADM Overview”](#) for more information.

Parameters

<code>inItem</code>	An ADM item.
<code>inNotifyProc</code>	callback with the following signature (see <code>ADMItem.h</code>): <pre>ADMItemNotifyProc)(ADMItemRef inItem, ADMNotifierRef inNotifier);</pre> Use <code>inNotifier</code> to use the functions in the ADM Notifier suite. You can use the ADM Notifier suite functions to determine the type of notification received. The <code>inNotifier</code> argument is passed to the ADM Notifier suite functions to indicate the event for which information is being requested.

Returns

None.

See also

[sADMItem->GetNotifyProc\(\)](#)

sADMItem->SetPicture()

Sets a picture based on an icon

```
void ASAPI (*SetPicture)(ADMItemRef inItem, ADMIconRef
inPicture);
```

Description

The **`SetPicture()`** function sets **`inItem`**'s picture. The picture is provided to ADM by you in the form of an ADM icon that was previously created/obtained by you.

NOTE: The same limited list of items applies here as in [sADMItem->SetPictureID\(\)](#).

Parameters

<code>inItem</code>	An ADM item.
<code>inPicture</code>	An ADM icon.

Returns

None

See also

[sADMItem->GetPicture\(\)](#)
[sADMItem->SetSelectedPictureID\(\)](#)
[sADMItem->SetPictureID\(\)](#)
[sADMItem->SetSelectedPicture\(\)](#)
[sADMItem->SetDisabledPicture\(\)](#)
[sADMItem->SetDisabledPictureID\(\)](#)

sADMItem->SetPictureID()

Set the item's picture ID

```
void WINAPI (*SetPictureID)(ADMItemRef inItem, ASInt32  
inPictureResID, const char* inPictureResName);
```

Description

The **SetPictureID()** function sets the picture to be displayed for **inItem**. The **inPictureResID** is the ID of a platform picture or icon resource.

Not all ADM Items have pictures. ADM Items for which this function sets a picture resource are:

```
#define kADMPictureCheckBoxType      "ADM Picture Check Box Button Type"  
#define kADMPicturePushButtonType   "ADM Picture Push Button Type"  
#define kADMPictureRadioButtonType  "ADM Picture Radio Button Type"  
#define kADMPictureStaticType       "ADM Picture Static Type"
```

A custom item type is based on a user item. This may or may not use a picture:

```
#define kADMUserType "ADM User Type"
```

Parameters

inItem	An ADM item.
inPictureResID	Resource ID for the picture.
inPictureResName	Picture resource name.

Returns

None.

See also

[sADMItem->GetPictureID\(\)](#)
[sADMItem->SetPicture\(\)](#)
[sADMItem->SetSelectedPictureID\(\)](#)
[sADMItem->SetSelectedPicture\(\)](#)

[sADMItem->SetDisabledPicture\(\)](#)
[sADMItem->SetDisabledPictureID\(\)](#)

sADMItem->SetPluginRef()

Set the plug-in that created an item

```
void ASAPI (*SetPluginRef)(ADMItemRef inItem, SPPluginRef
inPluginRef);
```

Description

The **SetPluginRef()** function sets the plug-in reference for **inItem**.

If you use the [sADMItem->Create\(\)](#) to make a new item, use this function to assign it an owning plug-in.

Parameters

inItem	An ADM item.
inPluginRef	Plug-in reference.

Returns

None.

See also

[sADMItem->GetPluginRef\(\)](#)

sADMItem->SetPopupDialog()

Set a popup dialog

```
void ASAPI (*SetPopupDialog)(ADMItemRef inItem, ASInt32
inPopupItemID, ADMDialogRef inDialog);
```

Description

The **SetPopupDialog()** is used to create a customized popup item. The function enables **inItem** to do its popup and to provide the dialog (**inDialog**) that will be displayed during the popped up state.

You need to call **SetPopupDialog()** to tell ADM what kind of a popup dialog should show up when the popup arrow is pressed. By default, a slider shows up. You can set a popup dial by placing the correct ID in **inPopupItemID** (see `ADMResource.h`).

If you want a custom popup dialog, you can create a dialog with the **kADMPopupControlDialogStyle** using [sADMDialog->Create\(\)](#) and pass the **ADMdialogref** in **SetPopupDialog()**. Note that it is your responsibility to destroy the custom **ADMdialogref**.

Parameters

inItem	An ADM item.
inPopupItemID	Pop-up dialog ID. (See <code>ADMResource.h</code> .)
inDialog	An ADM dialog.

Returns

None.

See also

[sADMItem->GetPopupDialog\(\)](#)

sADMItem->SetPrecision()

Set the precision of an item

```
void WINAPI (*SetPrecision)(ADMItemRef inItem, ASInt32  
inNumberOfDecimalPlaces);
```

Description

The **SetPrecision()** function sets the precision of **inItem**. **inNumberOfDecimalPlaces** is the number of digits that the user is allowed to enter following a decimal point. This function only applies to numeric edit text items.

If the user enters more numerals to the right of the decimal point than the item's precision allows, rounding is applied. If the precision is set to zero, decimals can be entered but they will be displayed as integers. A special precision value restricts the user to entering integers.

Parameters

inItem	An ADM item.
inNumberOfDecimalPlaces	Number of digits that the user is allowed to enter following a decimal point.

Returns

None.

See also

[sADMItem->GetPrecision\(\)](#)

sADMItem->SetRolloverPicture()

Set the rollover picture for an item

```
void ASAPI (*SetRolloverPicture)(ADMItemRef inItem, ADMIconRef
inPicture);
```

Description

The **SetRolloverPicture()** sets the rollover picture for **inItem**. A control is in a rollover state when the mouse hovers over it.

Parameters

inItem	An ADM item.
inPicture	An ADM icon.

Returns

None.

See also

[sADMItem->GetRolloverPicture\(\)](#)
[sADMItem->SetRolloverPictureID\(\)](#)
[sADMItem->SetHasRolloverProperty\(\)](#)
[sADMItem->SetInRolloverState\(\)](#)

sADMItem->SetRolloverPictureID()

Set the rollover picture ID of an item

```
void ASAPI (*SetRolloverPictureID)(ADMItemRef inItem, ASInt32
inPictureResID, const char* inPictureResName);
```

Description

The **SetRolloverPictureID()** sets the rollover picture ID for **inItem**. A control is in a rollover state when the mouse hovers over it.

Parameters

inItem	An ADM item.
inPictureResID	Resource ID for the picture.
inPictureResName	Picture resource name.

Returns

None.

See also

[sADMItem->GetRolloverPictureID\(\)](#)
[sADMItem->SetRolloverPicture\(\)](#)
[sADMItem->SetHasRolloverProperty\(\)](#)
[sADMItem->SetInRolloverState\(\)](#)

sADMItem->SetSelectedPicture()

Set the item's selected picture

```
void WINAPI (*SetSelectedPicture)(ADMItemRef inItem, ADMIconRef  
inPicture);
```

Description

The **SetSelectedPicture()** function sets the picture to be displayed for **inItem** when it is selected. You provide the picture of a previously created ADM icon.

NOTE: The same limited list of items applies here as in [sADMItem->SetSelectedPictureID\(\)](#).

Parameters

inItem	An ADM item.
inPicture	An ADM icon.

Returns

None.

See also

[sADMItem->GetSelectedPicture\(\)](#)
[sADMItem->SetPicture\(\)](#)
[sADMItem->SetSelectedPictureID\(\)](#)
[sADMItem->SetPictureID\(\)](#)
[sADMItem->SetDisabledPicture\(\)](#)
[sADMItem->SetDisabledPictureID\(\)](#)

sADMItem->SetSelectedPictureID()

Set the item's selected picture ID

```
void WINAPI (*SetSelectedPictureID)(ADMItemRef inItem, ASInt32  
inPictureResID, const char* inPictureResName);
```

Description

The **SetSelectedPictureID()** sets the picture to be displayed for **inItem** when it is selected. The **inPictureResID** is the ID of a platform picture or icon resource.

Not all ADM Items have pictures. ADM Items for which this function sets a picture resource are:

```
#define kADMPictureCheckBoxType    "ADM Picture Check Box Button Type"
#define kADMPicturePushButtonType  "ADM Picture Push Button Type"
#define kADMPictureRadioButtonType "ADM Picture Radio Button Type"
#define kADMPictureStaticType      "ADM Picture Static Type"
```

A custom item type is based on a user item. This may or may not use a picture:

```
#define kADMUserType    "ADM User Type"
```

If the item does not have a selected picture, ADM inverts the default picture to show that it is selected.

Parameters

inItem	An ADM item.
inPictureResID	Resource ID for the picture. If the item does not have a picture, it returns 0.
inPictureResName	Picture resource name.

Returns

None.

See also

```
sADMItem->GetSelectedPictureID()
sADMItem->SetPictureID()
sADMItem->SetSelectedPicture()
sADMItem->SetSelectedPictureID()
sADMItem->SetDisabledPicture()
sADMItem->SetDisabledPictureID()
```

sADMItem->SetSelectionRange()

Set the active text item's selection range

```
void ASAPI (*SetSelectionRange)(ADMItemRef inItem, ASInt32
inSelStart, ASInt32 inSelEnd);
```

Description

The **SetSelectionRange()** function sets the start and end positions of an active edit text item's selection. **inSelEnd** is equal to the last selected character and values greater than the item's text length are ignored.

This is for use with edit text items. The characters between the positions are selected. To set an insertion point only (no selection) set **inSelStart** equal to **inSelEnd**.

Parameters

inItem	An ADM item.
inSelStart	Beginning of selection. NOTE: inSelStart precedes the first selected character and thus might be 0.
inSelEnd	End of selection. Equal to the last selected character and can be no more than the item's text length.

Returns

None.

See also

[sADMItem->GetSelectionRange\(\)](#)

sADMItem->SetSmallIncrement()

Set the small increment value of an item

```
void ASAPI (*SetSmallIncrement)(ADMItemRef inItem, float  
inIncrement);
```

Description

The **SetSmallIncrement()** function sets the small increment value of **inItem**. The small increment is used by spinner, slider, and scroll bar items and is the amount to add or subtract from the item value when the user changes the value using the control (e.g., clicks in an arrow button with the mouse).

Parameters

inItem	An ADM item.
inIncrement	The small increment value for inItem .

Returns

None.

See also

[sADMItem->GetSmallIncrement\(\)](#)
[sADMItem->GetLargeIncrement\(\)](#)
[sADMItem->SetLargeIncrement\(\)](#)

sADMItem->SetText()

Set the item's text

```
void ASAPI (*SetText)(ADMItemRef inItem, const char* inText);
```

Description

The **SetText()** function sets **inItem**'s text to the indicated C string.

The use of an item's text depends on the item type. If used, it is generally displayed:

TABLE 14.2 Use of ADM Item Text

Item Type	Initialization
kADMFrameType	Text is the frame title.
kADMTextEditPopupType kADMSpinEditPopupType kADMSpinEditScrollingPopupType kADMPopupSpinEditControlType kADMSpinEditType kADMTextEditType kADMTextEditMultiLineType kADMTextEditScrollingPopupType	The text is the editable text.
kADMTextCheckBoxType	The text is displayed to the right of the check box.
kADMTextPushButtonType	The text is displayed within the push button.
kADMTextRadioButtonType	The text is displayed to the right of the radio button.
kADMTextStaticType kADMTextStaticMultiLineType	The text is displayed.
kADMUserType	Unknown.

TABLE 14.2 Use of ADM Item Text

Item Type	Initialization
kADMPicturePushButtonType	The text is unused.
kADMPictureRadioButtonType	
kADMPictureStaticType	
kADMPicturePushButtonType	
kADMPictureCheckBoxType	
kADMListBoxType	
kADMPopupListType	
kADMHierarchyListBoxType	
kADMPopupMenuType	
kADMScrollingPopupListType	
kADMResizeType	
kADMSliderType	
kADMScrollbarType	
kADMDialType	
kADMPopupControlType	
kADMPopupControlButtonType	
kADMPictureCheckBoxType	
kADMProgressbar	
kAADMChasingArrows	

Parameters

inItem	An ADM item.
inText	inItem 's text.

Returns

None.

See also

[sADMItem->GetText\(\)](#)

sADMItem->SetTextToFloatProc()

Sets an item's procedure for converting a text string to a floating point value

```
void ASAPI (*SetTextToFloatProc)(ADMItemRef inItem,  
ADMItemTextToFloatProc inProc);
```

Description

The **SetTextToFloatProc()** function sets the text to floating point conversion procedure for **inItem**. See [FloatToText](#) and [TextToFloat Functions](#) for more information.

Parameters

inItem	An ADM item.
inProc	A callback with the following signature (see <code>ADMItem.h</code>): ASBoolean ADMItemTextToFloatProc(ADMItemRef inItem, const char* inText, float* outValue);

Returns

None.

See also

[sADMItem->GetTextToFloatProc\(\)](#)
[sADMItem->DefaultTextToFloat\(\)](#)

sADMItem->SetTrackProc()

Set the ADM tracker function to use for the item

```
void ASAPI (*SetTrackProc)(ADMItemRef inItem, ADMItemTrackProc inTrackProc);
```

Description

The **SetTrackProc()** function defines event tracking function **inTrackProc** for **inItem**.

Within your tracking function you can use the ADM Tracker suite functions to access event information. The **ADMTrackerRef** argument is passed to the ADM Tracker suite functions to indicate the event for which information is being requested.

Your tracker function is called repeatedly until an end condition is signalled by [sADMTracker->Abort\(\)](#), called by your function or the default function. If the track function returns **true**, its item receives a notify event when the mouse is released. If it returns **false**, a notify event is not received—except in the case of text edits where **true/false** indicate whether or not a particular keystroke was handled.

An example of using a tracker function would be to have your function determine whether a modifier key is down, save the modifier state, and then call the default tracker. The default tracker would handle whether the item is to be notified. When called, your item notifier function would check the saved modifier state and act upon it.

See [Using Event Callbacks](#) in [Chapter 1, “ADM Overview”](#).

Parameters

inItem	An ADM item.
inTrackProc	A callback with the following signature (see <code>ADMItem.h</code>): ASBoolean ADMItemTrackProc(ADMItemRef inItem, ADMTrackerRef inTracker); inTracker is passed to the ADM Tracker suite functions to indicate the event for which information is being requested. This callback returns a boolean—if true , the item receives a notify event when the mouse is released; if false , a notify event will not be received. It always means whether or not to send notification <i>except</i> for keystroke events. In those cases it means whether or not the keystroke was handled.

Returns

None.

See also

[sADMTracker->Abort\(\)](#)

sADMItem->SetUnits()

Set the units of a text item

```
void ASAPI (*SetUnits)(ADMItemRef inItem, ADMUnits inUnits);
```

Description

The **SetUnits()** function sets the measurement units used for a text item.

Parameters

inItem	An ADM item.
inUnits	The units of measurement for inItem . Type: <code>ADMUnits</code> (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMItem->GetUnits\(\)](#)

sADMItem->SetUserData()

Set the user data pointer for an item

```
void ASAPI (*SetUserData)(ADMItemRef inItem, ADMUserData
inUserData);
```

Description

The **SetUserData()** function sets the 4-byte user value stored with **inItem**. It is automatically set by the [sADMItem->Create\(\)](#) function.

An ADM item's user data is independent of its dialog's data. To get the dialog's user data, get the item's dialog and then get the dialog's user data directly.

Parameters

inItem	An ADM item.
inUserData	The 4-byte user value stored with inItem . Type: ADMUserData (see ADMTypes.h)

Returns

None.

See also

[sADMItem->GetUserData\(\)](#)
[sADMItem->Create\(\)](#)

sADMItem->SetWantsFocus()

Set indicator that item wants focus

```
void ASAPI (*SetWantsFocus)(ADMItemRef inItem, ASBoolean
inWantsFocus);
```

Description

The **SetWantsFocus()** function sets an indicator that **inItem** wants focus. This function name is a bit of a misnomer—it really means that, if set, the item *can* accept focus, not that it “desires” it.

Third-party plug-in developers probably will never use this API. The only two known uses for it are:

- You create an ADM edit field and then use that to create a custom resource that will receive focus. (Not recommended.)
- You have an ADM picture push-button item type that you want to receive focus just like other controls do (on Windows). This is a more legitimate use of the API.

Currently the only controls that might see any effect from this API are the ones that already can normally receive focus (e.g., edit fields) and picture push-buttons (on Windows).

Parameters

inItem	An ADM item.
inWantsFocus	true indicates that inItem wants focus; false indicates otherwise.

Returns

None.

See also

[sADMItem->GetWantsFocus\(\)](#)

sADMItem->Show()

Hide or show an item

```
void ASAPI (*Show)(ADMItemRef inItem, ASBoolean inShow);
```

Description

The **Show()** function hides or shows **inItem**. Pass **true** to make an item visible and **false** to make it invisible.

Parameters

inItem	An ADM item.
inShow	If true , makes inItem visible; if false , makes it invisible.

Returns

None.

sADMItem->ShowUnits()

Show the units in value fields of an item

```
void ASAPI (*ShowUnits)(ADMItemRef inItem, ASBoolean inShow);
```

Description

The **ShowUnits()** function sets up the value fields to show the selected units for the fields.

Parameters

inItem	An ADM item.
inShow	If true , selected units for the field are show; if false , they are not.

Returns

None.

See also

[sADMItem->GetShowUnits\(\)](#)

sADMItem->Size()

Set the dimensions of an item

```
void ASAPI (*Size)(ADMItemRef inItem, ASInt32 inWidth, ASInt32
inHeight);
```

Description

The **Size()** function sets the bounds of **inItem** to the given **inWidth** and **inHeight**.

If the item is sized off the dialog window using this function, it will not be constrained.

Parameters

inItem	An ADM item.
inWidth	Width of inItem (in pixels).
inHeight	Height of inItem (in pixels).

Returns

None.

sADMItem->Update()

Force an update of an item

```
void ASAPI (*Update)(ADMItemRef inItem);
```

Description

The **Update()** function invalidates the bounds of **inItem** within the dialog and immediately updates its contents. The redraw will occur if **inItem**'s bounds rect is both visible and "dirty."

Parameters

inItem	An ADM item.
---------------	--------------

Returns

None.

See also[sADMItem->Invalidate\(\)](#)

sADMItem->WasPercentageChange()

Find out whether a value change was a percentage change

```
ASBoolean ASAPI (*WasPercentageChange)(ADMItemRef inItem);
```

Description

The **WasPercentageChange()** determines whether a value change in **inItem** was a percentage change. You would use it if you are, for instance, trying to keep track of a history of changes to an object's scale.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Normally, this function returns a **false** result. However, in a Notify proc, if the notification was the result of a text value change that included a percentage, such as **1in+10%** or **150%**, the return value is **true**. If **inItem** provides its own text-to-float proc (see [FloatToText and TextToFloat Functions](#)), the value is correct if the text-to-float proc called [sADMItem->DefaultTextToFloat\(\)](#) to do the actual parsing, otherwise it will be **false**.

See also[sADMItem->DefaultTextToFloat\(\)](#)

ADM Help Support

ADM has built-in support for ASHelp, a WinHelp-type help system. ASHelp uses WinHelp file definitions in a cross-platform fashion. Every item has a helpID and the system can operate in contextual fashion. For example, selecting **Command ?** in Macintosh or in **Alt + F1** in Windows lets you click an item and see that item's help resource. For plug-ins to support help files, there must be a Plugin Help location in the **PiPL** resource. The following functions are used with ASHelp.

NOTE: The Help API is deprecated in ADM V2.8. This does NOT include tool tips.

sADMItem->EnableTip()

Enable an item's tool tip string

```
void ASAPI (*EnableTip)(ADMItemRef inItem, ASBoolean
inEnable);
```

Description

The **EnableTip()** function enables or disables the tool tip of **inItem**. An enabled tool tip is displayed when the mouse is held over the item for a given time period. If an item's tool tip is disabled, no tip is displayed.

Parameters

inItem	An ADM item.
inEnable	If true , tool tip is enabled; if false , tool tip is disabled.

Returns

None.

See also

[sADMItem->IsTipEnabled\(\)](#)

sADMItem->GetHelpID()

Get the Help ID of an item

```
ASHelpID ASAPI (*GetHelpID)(ADMItemRef inItem);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **GetHelpID()** function gets the help ID of **inItem**.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

The help ID. Type: **ASHelpID** (See **ASHelp.h**)

See also

[sADMItem->SetHelpID\(\)](#)

sADMItem->GetTipString()

Get the tool tip string for an item

```
void ASAPI (*GetTipString)(ADMItemRef inItem, char*  
outTipString, ASInt32 inMaxLength);
```

Description

The **GetTipString()** function gets the tool tip of **inItem** in the supplied character buffer. The maximum number of characters that the buffer will hold is **inMaxLength**.

Parameters

inItem	An ADM item.
outTipString	Tool tip of inItem .
inMaxLength	Size of the already allocated buffer for outTipString .

Returns

None.

See also

[sADMItem->SetTipString\(\)](#)
[sADMItem->GetTipStringLength\(\)](#)

sADMItem->GetTipStringLength()

Get the length of an item's tool tip string

```
ASInt32 ASAPI (*GetTipStringLength)(ADMItemRef inItem);
```

Description

The **GetTipStringLength()** function returns the number of characters in the tool tip string of **inItem**.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

Number of characters in the tool tip string of **inItem**.

See also

[sADMItem->GetTipString\(\)](#)
[sADMItem->SetTipString\(\)](#)

sADMItem->Help()

Call the Help routine associated with an item

```
void ASAPI (*Help)(ADMItemRef inItem);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **Help()** function calls the ASHelp routine of **inItem**.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

None.

sADMItem->HideToolTip()

Hide an item's tool tip string

```
void ASAPI (*HideToolTip)(ADMItemRef inItem);
```

Description

The **HideToolTip()** function removes **inItem**'s displayed tool tip from the screen.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

None.

See also

[sADMItem->ShowToolTip\(\)](#)

sADMItem->IsTipEnabled()

Check if an item's tool tip string is enabled

```
ASBoolean ASAPI (*IsTipEnabled)(ADMItemRef inItem);
```

Description

The **IsTipEnabled()** function returns whether the tool tip of an item is enabled or disabled. If enabled, it returns **true**; if disabled, it returns **false**.

Parameters

inItem	An ADM item.
---------------	--------------

Returns

true if tool tip is enabled; **false** otherwise.

See also

[sADMItem->EnableTip\(\)](#)

sADMItem->SetHelpID()

Set the Help ID of an item

```
void ASAPI (*SetHelpID)(ADMItemRef inItem, ASHelpID inHelpID);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **SetHelpID()** function sets the help ID for **inItem**. The **inHelpID** is the resource ID for the ASHelp resource.

Parameters

inItem	An ADM item.
inHelpID	The help ID. Type: ASHelpID (See ASHelp.h)

Returns

None.

See also

[sADMItem->GetHelpID\(\)](#)

sADMItem->SetTipString()

Set the tool tip string for an item

```
void ASAPI (*SetTipString)(ADMItemRef inItem, const char*  
inTipString);
```

Description

The **SetTipString()** function sets the tool tip of **inItem** to the supplied C string.

Parameters

inItem	An ADM item.
---------------	--------------

inTipString	Tool tip of inItem .
--------------------	-----------------------------

Returns

None.

See also

[sADMItem->GetTipString\(\)](#)
[sADMItem->GetTipStringLength\(\)](#)

sADMItem->ShowToolTip()

Show an item's tool tip string

```
void ASAPI (*ShowToolTip)(ADMItemRef inItem, const ASPoint*  
inWhere);
```

Description

The **ShowToolTip()** function displays the tool tip for **inItem** as if the mouse were held over it.

Parameters

inItem	An ADM item.
inWhere	Position for display of the tool tip. Type: ASPoint (See ASTypes.h)

Returns

None.

See also

[sADMItem->HideToolTip\(\)](#)

15

The ADM List Suite

About the ADM List Suite

The ADM List suite allows you to access ADM List objects and ADM List entries. Since an ADM list is an extended property of a standard ADM item, this suite lacks many of the functions common to ADM objects; however, you can access the list's ADM Item and do common operations on it. Using functions in this suite you can initialize the list, and you can create, destroy, customize, and iterate through the entries of a list. The list suite is used in conjunction with the ADM Entry suite to further access list related information.

Accessing the Suite

The ADM List suite is referred to as:

```
#define kADMListSuite          "ADM List Suite"
```

with the version constant:

```
#define kADMListSuiteVersion2      2
```

NOTE: Determine the suite version number you are using by examining the `ADMList.h` header file.

The suite is acquired as follows:

```
ADMListSuite *sADMList;  
error = SSPBasic->AcquireSuite(kADMListSuite, kADMListSuiteVersion2,  
    &sADMList);  
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

ADM Lists and Entries

ADM lists are used by any ADM item that provides a list of choices, including list boxes, popup lists, popup menus, spin edit popups, and text edit popups. An ADM list is composed of ADM entries.

ADM lists do not have many standard properties, such as plug-in and bounds. Rather, these are defined using the ADM list's item. To access them, use the [sADMList->GetItem\(\)](#) function to get the item owning the list and then use the ADM Item suite functions (see [Chapter 14, "The ADM Item Suite"](#)) with the returned item reference.

ADM lists have special properties, such as a menu resource ID and a group of entries. ADM entries (see [Chapter 10, "The ADM Entry Suite"](#)) have additional properties,

including an index and a selected state. These entry properties are used by the ADM List suite to access the entries. The index is the position of the entry in list. The selected state indicates the user has selected the item (others may be selected in the case of a multi-select list).

ADM List Recipes

To get the ADM List object for an item, use [sADMItem->GetList\(\)](#):

```
ADMListRef theItemsList = sADMItem->GetList(theItem);
```

Once this is done you can use the ADM List and Entry suite functions to modify it.

To initialize a list, assign it a menu resource ID:

```
sADMList->SetMenuID(theItemsList, gPlugInRef, 16000, "Choices");
```

You can also create each entry with the [sADMList->InsertEntry\(\)](#) function followed by the [sADMEEntry->SetText\(\)](#) and [sADMEEntry->SetID\(\)](#) functions:

```
for (index = 0; index < kNumberEntries; index++) {
    char menuText[255];
    ADMEEntryRef entry = sADMList->InsertEntry(theItem, index);
    sBasic->GetIndexString(thePlugin, 16000, index, menuText, 255);
    sADMEEntry->SetText(entry, menuText);
    sADMEEntry->SetID(entry, index);
}
```

NOTE: List indices are 0-based.

To get the currently selected item of a single selection list, use the [sADMList->GetActiveEntry\(\)](#) function and then get the entry's index:

```
int GetListValue(ADMItem theListItem) {
    ADMList theList = sADMItem->GetList(theListItem);
    ADMEEntry theEntry = sADMList->GetActiveEntry(theItem);
    return sADMEEntry->GetID(theEntry);
}
```

To get each selected item in a multiple selection list, get the selection count and iterate through the selections:

```
int count = NumberOfSelectedEntries(theList);
for (index = 0; index < count; index++) {
    ADMEEntryRef entry = sADMList->IndexSelectedEntry(theList, index);
    doSomethingToSelectedEntry(theEntry);
}
```

Custom ADM Lists

You can customize an ADM List object just as you can customize other ADM items. This is done by defining one or more of the event handler functions. Because ADM lists are closely linked to ADM entries, the process is slightly different.

The ADM list doesn't actually have its own event handler functions. If you need to do something to the list as a whole in a handler, set the handler function for the list—for instance, to annotate the list, set the drawer function the list. These are assigned using the ADM Item suite.

If you need to change the behavior of the list at a lower level, you can set the handler functions of the list's entries; for instance, to change how each entry draws, you would set the drawer function for the list's entries. This is done at the list level, using the ADM List suite, and affects all the entries in a list. You cannot directly set a handler function for an individual entry; a custom handler function for a list must work for all its entries.

To use the default behavior for a list item, you use the ADM Item suite functions. To use the default behavior for a list entry, you use functions in the ADM Entry suite; not the ADM List suite.

ADM List Suite Functions

sADMList->FindEntry()

Get an entry by text

```
ADMLEntryRef ASAPI (*FindEntry)(ADMListRef inList, const char*
inText);
```

Description

The **FindEntry()** function gets a reference to the entry based on its text. ADM searches **inList** for an entry with text that matches **inText** and returns that entry. If no match is found, **NULL** is returned.

Parameters

inList	An ADM list.
inText	Entry text to search for.

Returns

The ADM entry with text that matches **inText**. If no match is found, **NULL** is returned.

sADMList->GetActiveEntry()

Get the selected entry of a list

```
ADMLEntryRef ASAPI (*GetActiveEntry)(ADMListRef inList);
```

Description

The **GetActiveEntry()** function returns a reference to the currently selected entry. If it is used on a list with multiple selections, it returns the first selection.

To get all the selected entries of multiple selection list, use the [sADMList->IndexSelectedEntry\(\)](#) function.

Parameters

inList	An ADM list.
---------------	--------------

Returns

Currently selected entry; if used on a list with multiple selections, returns the first selection.

See also

[sADMList->IndexSelectedEntry\(\)](#)

sADMList->GetDestroyProc()

Get the ADM destroy function being used for the list's entries

```
ADMEEntryDestroyProc WINAPI (*GetDestroyProc)(ADMListRef
inList);
```

Description

The **GetDestroyProc()** function returns the destroy function being used for **inList**'s entries.

Because ADM calls the entry's destroy function when it is removed from its list, you should not call the returned function directly.

Parameters

inList	An ADM list.
---------------	--------------

Returns

The Destroy proc for **inList**. If you have not called [sADMList->SetDestroyProc\(\)](#), returns **NULL** (not the default Destroy proc). Type: **ADMEEntryDestroyProc** (see `ADMList.h`)

See also

[sADMList->SetDestroyProc\(\)](#)

sADMList->GetDrawProc()

Get the ADM drawing function being used for the list's entries

```
ADMEEntryDrawProc ASAPI (*GetDrawProc)(ADMListRef inList);
```

Description

The **GetDrawProc()** function returns the drawing function being used for **inList**'s entries.

Rather than getting and calling the drawing function in this fashion, you are more likely to use the [sADMEEntry->DefaultDraw\(\)](#).

Parameters

inList	An ADM list.
---------------	--------------

Returns

The Drawer proc for **inList**. If you have not called [sADMList->SetDrawProc\(\)](#), returns **NULL** (not the default Drawer proc). Type: **ADMEEntryDrawProc** (see **ADMList.h**)

See also

[sADMEEntry->DefaultDraw\(\)](#)
[sADMList->SetDrawProc\(\)](#)

sADMList->GetEntry()

Get an entry by ID

```
ADMEEntryRef ASAPI (*GetEntry)(ADMListRef inList, ASInt32  
inEntryID);
```

Description

The **GetEntry()** function is used to get a reference to the entry with the indicated **inEntryID**. If no match is found, **NULL** is returned.

Each ADM Entry object has an ID, which can be obtained with the [sADMEEntry->GetID\(\)](#) function. If you keep this ID, you can pass it to this function at a later time and retrieve the item.

Parameters

inList	An ADM list.
inEntryID	ADM Entry object ID.

Returns

The entry with the indicated **inEntryID**; if no match is found, **NULL** is returned.

See also

[sADMLEntry->GetID\(\)](#)

sADMList->GetEntryHeight()

Get the height of a list's entries

```
ASInt32 ASAPI (*GetEntryHeight)(ADMListRef inList);
```

Description

The **GetEntryHeight()** function returns the height of **inList**'s entries. All entries have the same height.

Given this value and the local rectangle of the list, you can calculate the number of rows that will appear in the list.

Parameters

inList	An ADM list.
---------------	--------------

Returns

Height of **inList**'s entries.

See also

[sADMList->SetEntryHeight\(\)](#)

sADMList->GetEntryTextRect()

Get the edit text rectangle for a list

```
void ASAPI (*GetEntryTextRect)(ADMListRef inList, ASRect* outRect);
```

The **GetEntryTextRect()** function gets the location of the edit-in-place text item of **inList**.

Parameters

inList	An ADM list.
outRect	Location of the edit-in-place text item of inList . Type: ASRect (see ASTypes.h)

Returns

None.

See also[sADMList->SetEntryTextRect\(\)](#)

sADMList->GetEntryWidth()

Get the width of a column of entries

```
ASInt32 ASAPI (*GetEntryWidth)(ADMListRef inList);
```

Description

The **GetEntryWidth()** function returns the width of a column in **inList** if the list is part of a listbox item and has the style **kADMTileListBoxStyle**. For any other list style, this function returns the width of the single column of the list.

Parameters

inList	An ADM list.
---------------	--------------

Returns

Width of a column in **inList** if the list is part of a listbox item and has the style **kADMTileListBoxStyle**. For any other list style, returns the width of the single column of **inList**.

See also[sADMList->SetEntryWidth\(\)](#)

sADMList->GetInitProc()

Get the ADM init function to use for list entries

```
ADMEnterInitProc ASAPI (*GetInitProc)(ADMListRef inList);
```

Description

The **GetInitProc()** function gets the initialization function being used for **inList**'s entries.

You probably won't call the callback function directly. It is called by ADM each time you call the [sADMList->InsertEntry\(\)](#) function.

Parameters

inList	An ADM list.
---------------	--------------

Returns

The Init proc for **inList**. If you have not called [sADMList->SetInitProc\(\)](#), returns **NULL** (not the default Init proc). Type: **ADMEnterInitProc** (see **ADMList.h**)

See also[sADMList->SetInitProc\(\)](#)

sADMList->GetItem()

Get the item reference for a list

```
ADMItemRef ASAPI (*GetItem)(ADMListRef inList);
```

Description

The **GetItem()** function returns a reference to the ADM item to which **inList** belongs.

Since an ADM list is an extension of a standard ADM item, you need this reference to perform standard operations on an ADM list such as resizing it. Once you have this reference, you can use the ADM Item suite functions (see [Chapter 14, “The ADM Item Suite”](#)) to perform these operations.

Parameters

inList	An ADM list.
---------------	--------------

Returns

The ADM item to which **inList** belongs.

sADMList->GetMask()

Get the filter on events received by the entries' tracker

```
ADMActionMask ASAPI (*GetMask)(ADMListRef inList);
```

Description

The **GetMask()** function gets the mask that sets which events are tracked. (All entries have the same mask.)

Parameters

inList	An ADM list.
---------------	--------------

Returns

The mask. Type: **ADMActionMask** (see `ADMTypes.h`)

See also[sADMList->SetMask\(\)](#)

sADMList->GetMenuID()

Get the menu resource ID of a list

```
ASInt32 ASAPI (*GetMenuID)(ADMListRef inList);
```

Description

The **GetMenuID()** function returns the menu resource ID for **inList**. This is the standard platform menu resource used to initialize an ADM list. If **inList** was initialized manually by the plug-in, this will be 0.

Parameters

inList	An ADM list.
---------------	--------------

Returns

Resource ID for **inList**. ID of standard platform menu resource. If **inList** was initialized manually by the plug-in, will be 0.

See also[sADMList->SetMenuID\(\)](#)

sADMList->GetNotifierData()

Get the notifier data for a list entry

```
ADMUserData ASAPI (*GetNotifierData)(ADMListRef inList);
```

Description

The **GetNotifierData()** function returns the notification data of **inList**.

Parameters

inList	An ADM list.
---------------	--------------

Returns

Custom notification data. Type: **ADMUserData** (see `ADMTypes.h`)

See also[sADMList->SetNotifierData\(\)](#)

sADMList->GetNotifyProc()

Get the ADM notification function being used for entries

```
ADMLEntryNotifyProc ASAPI (*GetNotifyProc)(ADMListRef inList);
```

Description

The **GetNotifyProc()** function returns the notification function being used for **inList**'s entries.

Rather than getting and calling an entry's notification function directly, you are more likely to use [sADMLEntry->DefaultNotify\(\)](#).

Parameters

inList	An ADM list.
---------------	--------------

Returns

The notification function used for the entries of **inList**. If you have not called [sADMList->SetNotifyProc\(\)](#), returns **NULL** (not the default Notify proc). Type: **ADMLEntryNotifyProc** (see **ADMList.h**)

See also

[sADMList->SetNotifyProc\(\)](#)
[sADMLEntry->DefaultNotify\(\)](#)

sADMList->GetTrackProc()

Get the ADM tracker function being used for the list's entries

```
ADMDialogTrackProc WINAPI (*GetTrackProc)(ADMListRef inList);
```

Description

The **GetTrackProc()** function returns the event tracking function being used for a **inList**'s entries.

Rather than getting and calling a tracker function directly, you are more likely to use the [sADMLEntry->DefaultTrack\(\)](#) function.

Parameters

inList	An ADM list.
---------------	--------------

Returns

The tracker function used for the entries of **inList**. If you have not called [sADMList->SetTrackProc\(\)](#), returns **NULL** (not the default track proc). Type: **ADMDialogTrackProc** (see **ADMList.h**)

See also

[sADMList->SetTrackProc\(\)](#)

sADMList->GetUserData()

Get the user data value for a list

```
ADMUserData ASAPI (*GetUserData)(ADMListRef inList);
```

Description

The **GetUserData()** function returns the 4-byte user value stored with **inList**.

The meaning of the value is defined by the list's creator. It is likely a pointer to a data structure, for instance, the plug-in's globals. For some items, it might be a simple 4-byte type, such as a long or a fixed number.

An ADM list's user data is independent of its item's data.

Parameters

inList	An ADM list.
---------------	--------------

Returns

A 4-byte value that ADM keeps with **inList**. Type: **ADMUserData** (see **ADMTypes.h**)

See also

[sADMList->SetUserData\(\)](#)

sADMList->IndexEntry()

Get an entry by index

```
ADMLEntryRef ASAPI (*IndexEntry)(ADMListRef inList, ASInt32  
inIndex);
```

Description

The **IndexEntry()** function is used to get a reference to the entry at the indicated **inIndex**.

Using this function with the [sADMList->NumberOfEntries\(\)](#), you can iterate through all of a list's entries (see example).

Parameters

inList	An ADM list.
inIndex	The index of the entry for which a reference is requested.

Returns

The ADM entry at **inIndex**.

See also[sADMList->NumberOfEntries\(\)](#)**Example**

```

ADMListRef list;
ADMEEntryRef entry;
int i, count = sADMList->NumberOfEntries(list);

for ( i = 0; i < count; i++ ) {
    entry = sADMList->IndexEntry(list, i);
    // do something with the entry
}

```

sADMList->IndexSelectedEntry()

Get one of multiple selected entries of a list

```

ADMEEntryRef ASAPI (*IndexSelectedEntry)(ADMListRef inList,
ASInt32 inSelectionIndex);

```

Description

The **IndexSelectedEntry()** function is used to get a reference to one of several selected entries in a multiple selection list.

When used in conjunction with [sADMList->NumberOfSelectedEntries\(\)](#), you can iterate through all of a list's selected entries.

Parameters

inList	An ADM list.
inSelectionIndex	The 0-based index of the selected entry in inList for which a reference is requested.

Returns

A selected entry.

See also[sADMList->NumberOfSelectedEntries\(\)](#)**Example**

```

ADMListRef list;
ADMEEntryRef entry;
int i, count = sADMList->NumberOfSelectedEntries(list);

for (i = 0; i < count; i++) {
    entry = sADMList->IndexSelectedEntry(list, i);
    // do something with the selected entry
}

```



```
}
```

sADMList->InsertEntry()

Add an entry to a list

```
ADMLEntryRef WINAPI (*InsertEntry)(ADMListRef inList, ASInt32  
inIndex);
```

Description

The **InsertEntry()** function adds an ADM entry to **inList**. **inIndex** specifies where the entry is to be placed. The new entry is placed before entries with indices equal to or greater than **inIndex**.

This function allocates the entry and then calls your list entry init function, if you have specified one.

Parameters

inList	An ADM list.
inIndex	The index where to insert the new entry into inList .

Returns

A reference to the new entry.

See also

[sADMList->SetInitProc\(\)](#)
[sADMList->RemoveEntry\(\)](#)

sADMList->NumberOfEntries()

Get the number of entries in a list

```
ASInt32 WINAPI (*NumberOfEntries)(ADMListRef inList);
```

Description

The **NumberOfEntries()** function returns the number of entries in **inList**.

Used in conjunction with [sADMList->IndexEntry\(\)](#), you can iterate through all of a list's entries.

Parameters

inList	An ADM list.
---------------	--------------

Returns

The number of entries in **inList**.

See also[sADMList->IndexEntry\(\)](#)

sADMList->NumberOfSelectedEntries()

Get the number of selected entries in a list

```
ASInt32 ASAPI (*NumberOfSelectedEntries)(ADMListRef inList);
```

Description

The **NumberOfSelectedEntries()** function returns the number of selected entries in **inList**.

When used in conjunction with [sADMList->IndexSelectedEntry\(\)](#), you can iterate through all selected entries of a list.

Parameters

inList	An ADM list.
---------------	--------------

Returns

Number of selected entries in **inList**.

See also[sADMList->IndexSelectedEntry\(\)](#)

sADMList->PickEntry()

Select an entry at a specified point

```
ADMLEntryRef ASAPI (*PickEntry)(ADMListRef inList, const  
ASPoint* inPoint);
```

Description

The **PickEntry()** function selects the entry at **inPoint**. The point is given in **inList**'s coordinate space.

Parameters

inList	An ADM list.
inPoint	A point in inList . Type: ASPoint (see ASTypes.h)

Returns

The ADM entry at **inPoint** in **inList**.

sADMList->RemoveEntry()

Remove an entry from a list

```
void ASAPI (*RemoveEntry)(ADMListRef inList, ASInt32 inIndex);
```

Description

The **RemoveEntry()** function removes the entry at **inIndex** from the list. This function calls your list entry destroy function if you have specified one.

Parameters

inList	An ADM list.
inIndex	Index of the entry to remove.

Returns

None.

See also

[sADMList->InsertEntry\(\)](#)
[sADMList->SetDestroyProc\(\)](#)

sADMList->SelectByText()

Searches for and selects text

```
void ASAPI (*SelectByText)(ADMListRef inList, const char*  
inFindText);
```

Description

The **SelectByText()** function searches for and selects text.

Parameters

inList	An ADM list.
inFindText	Text to search for and select.

Returns

None.

sADMList->SetBackgroundColor()

Set the background color

```
void ASAPI (*SetBackgroundColor)(ADMListRef inList, ADMColor  
inColor);
```

Description

The **SetBackgroundColor()** function sets the background color of **inList**. Default behavior is restored by specifying **kADMDummyColor**.

Parameters

inList	An ADM list.
inColor	The color to which to set the background of inList . Type: ADMColor (see ADMTypes.h)

Returns

None.

sADMList->SetDestroyProc()

Set the ADM destroy function to use for the list's entries

```
void ASAPI (*SetDestroyProc)(ADMListRef inList,
    ADMEEntryDestroyProc inDestroyProc);
```

Description

The **SetDestroyProc()** function assigns destroy function **inDestroyProc** to **inList**'s entries. This callback is called when [sADMList->RemoveEntry\(\)](#) is called for an entry. The callback should free memory and other resources that may have been allocated in the entry initialization function.

ADM destroys the entry, so you do not need to call a default destroy function from within your function.

Parameters

inList	An ADM list.
inDestroyProc	A callback with the following signature (see ADMItem.h): ADMEEntryDestroyProc(ADMEEntryRef inEntry);

Returns

None.

See also

[sADMList->GetDestroyProc\(\)](#)

sADMList->SetDrawProc()

Set the ADM drawing function to use for the list's entries

```
void ASAPI (*SetDrawProc)(ADMListRef inList, ADMEEntryDrawProc  
inDrawProc);
```

Description

The **SetDrawProc()** function defines drawing function **inDrawProc** for **inList**'s entries.

Within your drawing function, you can use the ADM Drawer suite functions to perform standard image operations such as drawing lines and pictures. The **inDrawer** argument is passed to the ADM Drawer suite functions to indicate where imaging occurs.

To call the default drawing function for an entry, use [sADMEEntry->DefaultDraw\(\)](#).

See [Using Event Callbacks](#) in [Chapter 1, "ADM Overview"](#) for more information.

Parameters

inList	An ADM list.
inDrawProc	A callback with the following signature (see <code>ADMItem.h</code>): ADMEEntryDrawProc(ADMEEntryRef inEntry, ADMDrawerRef inDrawer); Within a drawing function you can use the ADM Drawer suite functions to perform standard image operations such as drawing lines and pictures. The inDrawer argument is passed to the ADM Drawer suite functions to indicate where imaging occurs.

Returns

None.

sADMList->SetEntryHeight()

Set the height of a list's entries

```
void ASAPI (*SetEntryHeight)(ADMListRef inList, ASInt32  
inHeight);
```

Description

The **SetEntryHeight()** function sets the height of a **inList**'s entries and, indirectly, the number of rows that will appear in the list. All entries have the same height.

Parameters

inList	An ADM list.
---------------	--------------

inHeight	Height of inList 's entries.
-----------------	-------------------------------------

Returns

None.

See also[sADMList->GetEntryHeight\(\)](#)**sADMList->SetEntryTextRect()**

Set the edit text rectangle for a list

```
void ASAPI (*SetEntryTextRect)(ADMListRef inList, const
ASRect* inRect);
```

Description

The **SetEntryTextRect()** function sets the size of the text rectangle for **inList**. This is the rectangle within which text is drawn.

Parameters

inList	An ADM list.
inRect	Location for the edit-in-place text item of inList . Type: ASRect (see ASTypes.h)

Returns

None.

See also[sADMList->GetEntryTextRect\(\)](#)**sADMList->SetEntryWidth()**

Set the width of a column of entries

```
void ASAPI (*SetEntryWidth)(ADMListRef inList, ASInt32
inWidth);
```

Description

The **SetEntryWidth()** function sets the width of a column if **inList** is part of a listbox item and has the style **kADMTileListBoxStyle**. For any other list type or style, this function sets the width of the list entries' local and bounds rect.

For listbox items of style **kADMTileListBoxStyle**, the listbox is made up of rows and columns. This function sets the width of a column and indirectly sets the number of columns to appear. It does not create rows for other list types and styles, but may affect the list's appearance.

Parameters

inList	An ADM list.
inWidth	Width of a column in inList if the list is part of a listbox item and has the style kADMTileListBoxStyle . For any other list type or style, sets the width of the entries' local and bounds rect.

Returns

None.

See also

[sADMList->GetEntryWidth\(\)](#)

sADMList->SetInitProc()

Set the ADM init function to use for entries

```
void ASAPI (*SetInitProc)(ADMListRef inList, ADMEEntryInitProc  
inInitProc);
```

Description

The **SetInitProc()** function defines an initialization function **inInitProc** for **inList**. This callback is called each time an entry is created within the indicated list.

Within your initialization function, you can allocate memory or other resources and you can initialize variables. You do not need to allocate the entry itself, as ADM handles this.

NOTE: The list itself does not have an initialization function. This is handled at the ADM item level.

Parameters

inList	An ADM list.
inInitProc	A callback with the following signature (see <code>ADMItem.h</code>): ADMEEntryInitProc(ADMEEntryRef inEntry);

Returns

None.

See also

[sADMList->GetInitProc\(\)](#)

sADMList->SetMask()

Set a filter on events received by the entries' tracker

```
void ASAPI (*SetMask)(ADMListRef inList, ADMActionMask
inActionMask);
```

Description

The **SetMask()** function sets the mask for tracking the events associated with the entries of **inList**. (All entries have the same mask.)

Parameters

inList	An ADM list.
inActionMask	The mask. Type: ADMActionMask (see ADMTypes.h)

Returns

None.

See also

[sADMList->GetMask\(\)](#)

sADMList->SetMenuID()

Set the menu resource ID of a list

```
void ASAPI (*SetMenuID)(ADMListRef inList, SPPluginRef
inMenuResPlugin, ASInt32 inMenuResID, const char*
inMenuResName);
```

Description

The **SetMenuID()** function sets the menu resource ID of a list. Setting the menu ID causes ADM to read the resource and map the platform menu items to ADM list entries. **menuResID** is an ID of a standard platform menu resource.

If the list has already been assigned a resource ID and **SetMenuID()** is called again, the existing list will be disposed before the new one is added.

Parameters

inList	An ADM list.
inMenuResPlugin	Plug-in reference.
inMenuResID	Resource ID for inList . ID of standard platform menu resource.
inMenuResName	List resource name.

Returns

None.

See also

[sADMList->GetMenuID\(\)](#)
[sADMHierarchyList->GetMenuID\(\)](#)

sADMList->SetNotifierData()

Set notifier data for a list entry

```
void WINAPI (*SetNotifierData)(ADMListRef inList, ADMUserData  
inUserData);
```

Description

The **SetNotifierData()** function sets the notification data of **inList**, if any. This is used for custom notification procedures.

Parameters

inList	An ADM list.
inUserData	Custom notification data. Type: ADMUserData (see ADMTypes.h)

Returns

None.

See also

[sADMList->GetNotifierData\(\)](#)

sADMList->SetNotifyProc()

Set the ADM notification function to use for the list's entries

```
void WINAPI (*SetNotifyProc)(ADMListRef inList,  
ADMEEntryNotifyProc inNotifyProc);
```

Description

The **SetNotifyProc()** function assigns event notification function **inNotifyProc** to **inList**'s entries. The notification is sent after a mouse-up event occurs on the entry.

Within your notification function you can use the ADM Notifier suite functions to determine the type of notification was received. The **inNotifier** argument is passed to the ADM Notifier suite functions to indicate the event for which information is being requested.

See [Using Event Callbacks](#) in [Chapter 1, "ADM Overview"](#) for more information.

Parameters

inList	An ADM list.
inNotifyProc	<p>A callback with the following signature (see <code>ADMItem.h</code>):</p> <pre>ADMEnterNotifyProc(ADMEnterRef inEntry, ADMNotifierRef inNotifier);</pre> <p>Within your notification function you can use the ADM Notifier suite functions to determine the type of notification received. The inNotifier argument is passed to the ADM Notifier suite functions to indicate the event for which information is being requested.</p>

Returns

None.

See also

[sADMList->GetNotifyProc\(\)](#)

sADMList->SetTrackProc()

Set the ADM tracker function to use for the list's entries

```
void ASAPI (*SetTrackProc)(ADMListRef inList,  
ADMEnterTrackProc inTrackProc);
```

Description

The **SetTrackProc()** function defines event tracking function **inTrackProc** for **inList**.

Within your track function you can use the ADM Tracker suite functions to access event information. The **inTracker** argument is passed to the ADM Tracker suite functions to indicate the event for which information is being requested.

See [Using Event Callbacks](#) in [Chapter 1, "ADM Overview"](#) for more information.

Parameters

inList	An ADM list.
---------------	--------------

inTrackProc	<p>A callback with the following signature (see <code>ADMItem.h</code>):</p> <pre>ADMEEntryTrackProc(ADMEEntryRef inEntry, ADMTrackerRef inTracker);</pre> <p>Within your track function you can use the ADM Tracker suite functions to access event information. The inTracker argument is passed to the ADM Tracker suite functions to indicate the event for which information is being requested. Your function should return true if ADM should call the entry's notifier function or false if it shouldn't.</p>
--------------------	---

Returns

None

See also

[sADMList->GetTrackProc\(\)](#)

sADMList->SetUserData()

Set the user data value for a list

```
void WINAPI (*SetUserData)(ADMListRef inList, ADMUserData  
inUserData);
```

Description

The **SetUserData()** function sets the 4-byte user value stored with the list.

An ADM Item's user data is independent of its item's data. To get the item's user data, get the list's item and then get the item's user data using the list suite functions.

Parameters

inList	An ADM list.
inUserData	A 4-byte value that ADM keeps with inList . Type: ADMUserData (see <code>ADMTypes.h</code>)

Returns

None.

See also

[sADMList->GetUserData\(\)](#)

16

The ADM List Entry Suite

About the ADM List Entry Suite

An ADM List Entry object is a member of an ADM Hierarchical List object. It acts exactly like an ADM Entry, except that it has the possibility of containing child lists and entries. Since an ADM list entry is an extended property of a standard ADM Item, you can access the list entry's ADM item and perform operations on it. Using functions in this suite you can create, destroy, customize, and iterate through the entries of a hierarchical list. The List Entry suite is used in conjunction with the ADM Hierarchical List suite to further access list related information.

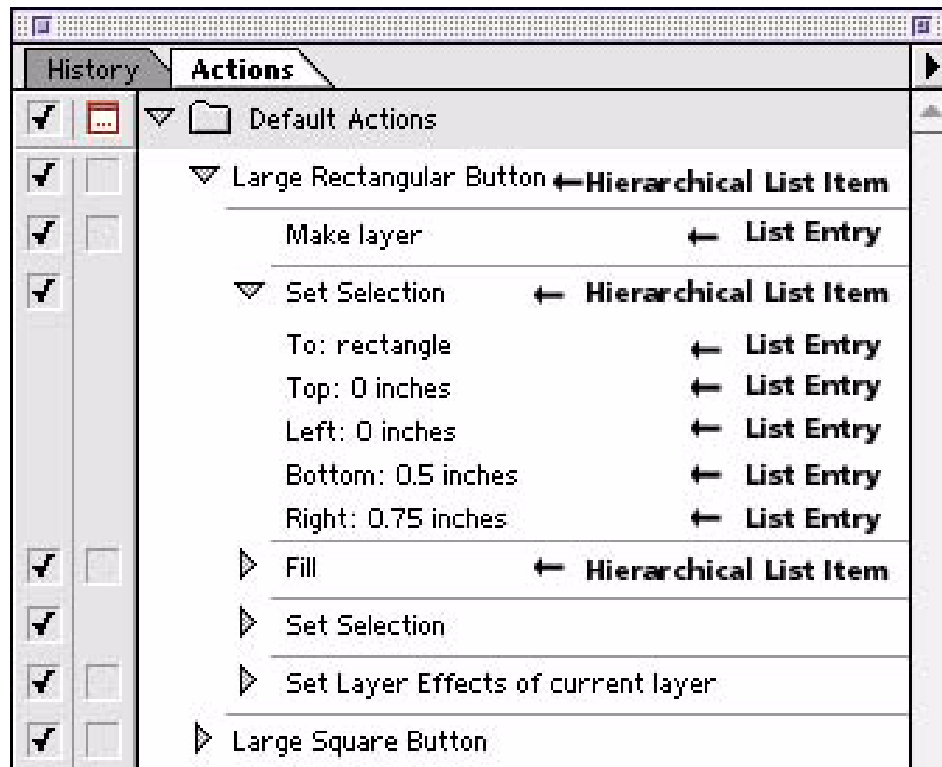


FIGURE 16.1 An ADM Hierarchy List

Accessing the Suite

The ADM List Entry suite is referred to as:

```
#define kADMListEntrySuite    "ADM List Entry Suite"
```

with the version constant:

```
#define kADMListEntrySuiteVersion2 2
```

NOTE: Determine the suite version number you are using by examining the `ADMListEntry.h` header file.

The suite is acquired as follows:

```
ADMListSuite *sADMListEntry;
error = SSPBasic->AcquireSuite(kADMListEntrySuite,
    kADMListEntrySuiteVersion2, &sADMListEntry);
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

ADM List Objects and Entries

ADM Hierarchy List Objects and List Entries

ADM List objects are used by any ADM Item object that provides a list of choices, including list boxes, popup lists, popup menus, spin edit popups, and text edit popups. An ADM list is composed of ADM entries. In similar fashion, an ADM Hierarchy List object is composed of ADM List Entry objects, which themselves may be hierarchical lists. See [Figure 16.1](#).

ADM hierarchy lists and list entries do not have many standard properties, such as plug-in and bounds. Rather, these are defined using the ADM hierarchy list's item. To access them, use `sADMListEntry->GetItem()` to get the item owning the hierarchy list and then use the ADM Item suite functions (see [Chapter 14, "The ADM Item Suite"](#)) with the returned item reference.

ADM lists and hierarchy lists have special properties, such as a menu resource ID and a group of entries. ADM entries and list entries have other additional properties, including an index and a selected state. These entry properties are used by the ADM List Entry suite to access the entries. The index is the position of the entry in the list. The selected state indicates the user has selected the item (others may be selected in the case of a multi-select list).

ADM List Entry Suite Functions

sADMListEntry->AbortTimer()

Abort timer

```
void ASAPI (*AbortTimer)(ADMListEntryRef inEntry, ADMTimerRef
inTimer);
```

Description

The **AbortTimer()** function aborts a timer procedure. It is used if the event specified by the **inAbortMask** in [sADMListEntry->CreateTimer\(\)](#) occurs or if you destroy your dialog before your timer expires.

Parameters

inEntry	An ADM list entry.
inTimer	An ADM timer.

Returns

[sADMListEntry->CreateTimer\(\)](#)

sADMListEntry->Activate()

Make an ADM list entry active or inactive

```
void ASAPI (*Activate)(ADMListEntryRef inEntry, ASBoolean  
inActivate);
```

Description

The **Activate()** function activates a list entry. Pass **true** to activate the list entry, **false** to inactivate it. Activates/deactivates the associated list item.

NOTE: A list entry's active and select states are the same.

Parameters

inEntry	An ADM list entry.
inActivate	If true , the entry is activated; if false , it is inactivated.

Returns

None.

See also

[sADMListEntry->IsActive\(\)](#)

sADMListEntry->AreChildrenSelected()

Determine whether the children of an ADM list entry are selected

```
ASBoolean ASAPI (*AreChildrenSelected)(ADMListEntryRef  
inEntry);
```

Description

The **AreChildrenSelected()** function determines whether any of the children of **inEntry** are selected.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if a child or children of **inEntry** are active; **false** otherwise.

See also

[sADMListEntry->Select\(\)](#)
[sADMListEntry->IsSelected\(\)](#)

sADMListEntry->Check()

Check an ADM list entry

```
void ASAPI (*Check)(ADMListEntryRef inEntry, ASBoolean
inCheck);
```

Description

The **Check()** function places a check mark next to **inEntry**. Pass **true** to check **inEntry**, **false** to have no check. By default, only menu lists can have checked entries.

NOTE: This state is valid for list boxes, though unused. If you are implementing a custom drawer for a list's entries, you could use this value.

Parameters

inEntry	An ADM list entry.
inCheck	If true , the entry has a check mark; if false , it does not have a check mark.

Returns

None.

See also

[sADMListEntry->IsChecked\(\)](#)

sADMListEntry->Create()

Create an ADM hierarchy list entry

```
ADMListEntryRef ASAPI (*Create)(ADMHierarchyListRef inList);
```


Description

The **Create()** function makes a new ADM hierarchy list entry in **inList**. The ID of the list entry is 0 if you do not explicitly set it with the [sADMListEntry->SetID\(\)](#) function. Use [sADMListEntry->Destroy\(\)](#) to dispose of the entry.

Parameters

inList	An ADM list entry.
---------------	--------------------

Returns

The new **ADMListEntryRef**.

See also

[sADMListEntry->SetID\(\)](#)
[sADMListEntry->Destroy\(\)](#)

sADMListEntry->CreateChildList()

Create a child list for a list entry

```
ADMHierarchyListRef ASAPI (*CreateChildList)(ADMListEntryRef
inEntry);
```

Description

The **CreateChildList()** function creates a child list for **inEntry**. Once the child list is created, any of the ADM Hierarchy List suite's functions may be applied to the it.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

The new child list.

See also

[sADMListEntry->DeleteChildList\(\)](#)

sADMListEntry->CreateTimer()

Create a timer

```
ADMTimerRef ASAPI (*CreateTimer)(ADMListEntryRef inEntry,
ASUInt32 inMilliseconds, ADMActionMask inAbortMask,
ADMListEntryTimerProc inTimerProc, ADMListEntryTimerAbortProc
inAbortProc, ASInt32 inOptions);
```

Description

The **CreateTimer()** function creates a timer for measuring time between events. Time is kept in milliseconds, with a user-supplied **inTimerProc** and **inTimerAbortProc**. If the delay succeeds (i.e., not aborted) then the **inTimerProc** will be executed. If the action specified by the **inAbortMask** occurs, **inTimerAbortProc** is called.

Parameters

inEntry	An ADM list entry.
inMilliseconds	Number of milliseconds in the delay.
inAbortMask	Actions that cause a timer abort. If an action occurs, inTimerAbortProc is called. Maskable actions are listed in <code>ADMTracker.h</code> . Type: ADMActionMask (see <code>ADMTypes.h</code>)
inTimerProc	A callback with the following signature (see <code>ADMListEntry.h</code>): ASBoolean ADMListEntryTimerProc(ADMListEntryRef inEntry, ADMTimerRef inTimer); Executed if the delay succeeds. Returns a boolean. If it returns true , then inTimerProc will be called again after inMilliseconds . If it returns false then inTimerProc will no longer be called.
inTimerAbortProc	A callback with the following signature (see <code>ADMListEntry.h</code>): ADMListEntryTimerAbortProc(ADMListEntryRef inEntry, ADMTimerRef inTimer, ADMAction inAbortAction); Called if an action specified by inAbortMask occurs. The values for inAbortAction are of type ADMAction and are listed in <code>ADMTracker.h</code> .
inOptions	Currently unused. Always pass 0 (zero).

Returns

The new **ADMTimerRef**.

See also

[sADMListEntry->AbortTimer\(\)](#)

sADMListEntry->DefaultDraw()

Call an ADM list entry's default drawing function

```
void WINAPI (*DefaultDraw)(ADMListEntryRef inEntry,  
ADMDrawerRef inDrawer);
```

Description

The **DefaultDraw()** function calls the list entry's current default draw function from within your custom entry draw function. The arguments passed to the custom function are passed through to the **DefaultDraw()** call.

You will most likely call the default drawing routine within a custom draw function to get the basic appearance of the entry. Your draw function would then add to the entry's appearance. If you completely change the appearance of an entry, you should not call this function.

Set your custom drawer function using [sADMHierarchyList->SetDrawProc\(\)](#) or [sADMHierarchyList->SetDrawProcRecursive\(\)](#).

Parameters

inEntry	An ADM list entry.
inDrawer	An ADM drawer.

Returns

None.

See also

[sADMHierarchyList->SetDrawProc\(\)](#)
[sADMHierarchyList->SetDrawProcRecursive\(\)](#)

Example

```
void doNothingDrawHandler(ADMListEntryRef inEntry, ADMDrawerRef  
inDrawer) {  
    sADMListEntry->DefaultDraw(inEntry, inDrawer);  
}
```

sADMListEntry->DefaultNotify()Call an ADM list entry's default
notification function

```
void WINAPI (*DefaultNotify)(ADMListEntryRef inEntry,  
ADMNotifierRef inNotifier);
```

Description

The **DefaultNotify()** function calls the default notification function of the entry. Use this within a custom notification callback function. The arguments passed to the custom function are passed through to the **DefaultNotify()** call.

You will always call the default notification function for an entry to get standard behaviors.

Set your custom notifier function using [sADMHierarchyList->SetNotifyProc\(\)](#) or [sADMHierarchyList->SetNotifyProcRecursive\(\)](#).

Parameters

inEntry	An ADM list entry.
inNotifier	An ADM notifier.

Returns

None.

See also

[sADMHierarchyList->SetNotifyProc\(\)](#)
[sADMHierarchyList->SetNotifyProcRecursive\(\)](#)

Example

```
void doNothingNotificationHandler(ADMListEntryRef inEntry,
    ADMNotifierRef inNotifier) {
    sADMListEntry->DefaultNotify(inEntry, inNotifier);
    // Custom behavior would go here...
}
```

sADMListEntry->DefaultTrack() Call an ADM list entry's default tracking function

```
ASBoolean ASAPI (*DefaultTrack)(ADMListEntryRef inEntry,
    ADMTrackerRef inTracker);
```

Description

The **DefaultTrack()** function calls the default tracking function of the entry. Use it within a custom tracker callback function. The arguments passed to the custom function are passed through to the **DefaultTrack()** call.

The default tracker function handles entry selection, including multiple selections. A custom tracker would be used, for instance, to determine where the mouse-down event occurred so that you could toggle a picture.

You set your custom tracker function using [sADMHierarchyList->SetTrackProc\(\)](#) or [sADMHierarchyList->SetTrackProcRecursive\(\)](#).

Parameters

inEntry	An ADM list entry.
inTracker	An ADM tracker.

Returns

Boolean indicating results of the operation.

See also

[sADMHierarchyList->SetTrackProc\(\)](#)
[sADMHierarchyList->SetTrackProcRecursive\(\)](#)

Example

```
ASBoolean doNothingTrackHandler(ADMListEntryRef entry, ADMTrackerRef
tracker) {
    return sADMListEntry->DefaultTrack(inEntry, inTracker);
}
```

sADMListEntry->DeleteChildList()

Delete a child list for a list entry

```
void ASAPI (*DeleteChildList)(ADMListEntryRef inEntry);
```

Description

The **DeleteChildList()** function deletes a child list for **inEntry**.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

None.

See also

[sADMListEntry->CreateChildList\(\)](#)

sADMListEntry->Destroy()

Destroy an ADM hierarchy list entry

```
void ASAPI (*Destroy)(ADMListEntryRef inEntry);
```

Description

The **Destroy()** function removes **inEntry** from its list. If you have used [sADMHierarchyList->SetDestroyProc\(\)](#) or [sADMHierarchyList-](#)

[>SetDestroyProcRecursive\(\)](#) to give the entry a custom destroy function, your function will be triggered by this call.

ADM automatically destroys all entries in a list when the ADM dialog is destroyed. Use this function if you are creating and disposing of entries dynamically in response to user actions.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

None.

See also

[sADMHierarchyList->SetDestroyProc\(\)](#)
[sADMHierarchyList->SetDestroyProcRecursive\(\)](#)

sADMListEntry->Enable()

Enable or disable an ADM list entry

```
void ASAPI (*Enable)(ADMListEntryRef inEntry, ASBoolean
inEnable);
```

Description

The **Enable()** function enables or disables **inEntry**. An enabled entry can be selected by the user. A disabled entry is unusable and by default appears with grayed text and a grayed icon if it has one.

Parameters

inEntry	An ADM list entry.
inEnable	If true , the entry is enabled; if false , it is disabled.

Returns

None.

sADMListEntry->EnableChildSelection()

Sets whether or not the list entries of a child list are selectable

```
void ASAPI (*EnableChildSelection)(ADMListEntryRef inEntry,
ASBoolean inFlag);
```

Description

The **EnableChildSelection()** function sets whether or not the entries of a child list are selectable.

Parameters

inEntry	An ADM list entry.
inFlag	If true , child list entries are selectable; if false , they are not.

Returns

None.

sADMListEntry->ExpandHierarchy()

Expand a list for a given list entry

```
void ASAPI (*ExpandHierarchy)(ADMListEntryRef inEntry,
ASBoolean inFlag);
```

Description

The **ExpandHierarchy()** function expands a hierarchy list for an **inEntry**. When a list entry's hierarchy is expanded, all child entries are visible.

Parameters

inEntry	An ADM list entry.
inFlag	If true , inEntry 's hierarchy list is expanded; if false , list is not expanded.

Returns

None.

sADMListEntry->GetBoundsRect()

Get the absolute position and size of a list entry

```
void ASAPI (*GetBoundsRect)(ADMListEntryRef inEntry, ASRect*
outBoundsRect);
```

Description

The **GetBoundsRect()** function gets the current size and position of **inEntry** in its containing dialog's coordinate space.

Parameters

inEntry	An ADM list entry.
outBoundsRect	Current size and position of inEntry in its containing dialog's coordinate space. Type: ASRect (see ASTypes.h)

Returns

None.

sADMListEntry->GetChildList()

Get the child list for a list entry

```
ADMHierarchyListRef ASAPI (*GetItem)(ADMListEntryRef inEntry);
```

Description

The **GetItem()** function gets a reference to **inEntry**'s child list. Once the child list is obtained, any of the ADM Hierarchy List suite's functions may be applied to the it. Only one child list can exist per **inEntry**.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

An ADM list entry.

sADMListEntry->GetDisabledPicture()

Get the list entry disabled picture

```
ADMIconRef ASAPI (*GetDisabledPicture)(ADMListEntryRef  
inEntry);
```

Description

The **GetDisabledPicture()** function returns the picture that is set to display when **inEntry** is disabled.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

An ADM icon.

See also

[sADMListEntry->SetDisabledPicture\(\)](#)
[sADMListEntry->GetSelectedPicture\(\)](#)
[sADMListEntry->GetSelectedPictureID\(\)](#)
[sADMListEntry->GetPicture\(\)](#)
[sADMListEntry->GetPictureID\(\)](#)
[sADMListEntry->GetDisabledPictureID\(\)](#)

sADMListEntry->GetDisabledPictureID()

Get the list entry's disabled picture ID

```
void ASAPI (*GetDisabledPictureID)(ADMListEntryRef inEntry,  
ASInt32* outPictureResID, const char** outPictureResName);
```

Description

The **GetDisabledPictureID()** function gets the resource ID of the picture that is set to be displayed for **inEntry** when it is disabled. If the entry does not have a disabled picture, **outPictureResID** returns 0.

Parameters

inEntry	An ADM list entry.
outPictureResID	Resource ID for the picture. If the entry does not have a disabled picture, returns 0.
outPictureResName	Picture resource name.

Returns

None.

See also

[sADMListEntry->SetDisabledPictureID\(\)](#)
[sADMListEntry->GetSelectedPicture\(\)](#)
[sADMListEntry->GetSelectedPictureID\(\)](#)
[sADMListEntry->GetPicture\(\)](#)
[sADMListEntry->GetPictureID\(\)](#)
[sADMListEntry->GetDisabledPicture\(\)](#)

sADMListEntry->GetEntryItem()

Get the list entry's item

```
ADMListEntryRef ASAPI (*GetEntryItem)(ADMListEntryRef inEntry);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **GetEntryItem()** gets **inEntry**'s item.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

inEntry's item.

sADMListEntry->GetExpandArrowLocalRect() Get location of expand/collapse arrow

```
void ASAPI (*GetExpandArrowLocalRect)(ADMListEntryRef inEntry,
ASRect* outLocalRect);
```

Description

The **GetExpandArrowLocalRect()** function gets the location of the expand/collapse arrow within the bounds rectangle so you can draw to the right of it.

Parameters

inEntry	An ADM list entry.
outLocalRect	Location of the expand/collapse arrow within the bounds rectangle. Type: ASRect (see ASTypes.h)

Returns

None.

sADMListEntry->GetHierarchyDepth() Get the depth of a hierarchy list entry

```
ASInt32 ASAPI (*GetHierarchyDepth)(ADMListEntryRef inEntry);
```

Description

The **GetHierarchyDepth()** function gets the depth of **inEntry**. It returns how far down the list the entry is located.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

How far down the hierarchy list **inEntry** is located (0-based).

See also

[sADMListEntry->GetVisualHierarchyDepth\(\)](#)

sADMListEntry->GetID()

Get the ID of an ADM list entry

```
ASInt32 ASAPI (*GetID)(ADMListEntryRef inEntry);
```

Description

The **GetID()** function returns the ID of **inEntry**. When ADM creates list entries using a menu resource, it sets the initial ID of each entry to its index + 1. If you create the entry, it initially has an ID of 0.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

ID of **inEntry**.

See also

[sADMListEntry->SetID\(\)](#)

sADMListEntry->GetIndex()

Get the index of a list entry

```
ASInt32 ASAPI (*GetIndex)(ADMListEntryRef inEntry);
```

Description

Use this to get the index, or position, of a hierarchy list entry.

The **GetIndex()** function gets the index, or position, of **inEntry** in the hierarchy list.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

The index of **inEntry** in a hierarchy list.

See also

[sADMListEntry->GetIndex\(\)](#)

sADMListEntry->GetItem()

Get the item reference for a list entry

```
ADMItemRef ASAPI (*GetItem)(ADMListEntryRef inEntry);
```

Description

The **GetItem()** function gets a reference to the ADM item to which the list entry belongs.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

The ADM item to which the list entry belongs.

sADMListEntry->GetList()

Get the parent hierarchy list for a given list entry

```
ADMHierarchyListRef ASAPI (*GetList)(ADMListEntryRef inEntry);
```

Description

The **GetList()** function returns a reference to **inEntry**'s containing ADM List object. Once obtained, the ADM Hierarchy List suite functions can be used to access the list (see [Chapter 11, "The ADM Hierarchy List Suite"](#)).

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

An ADM hierarchy list.

sADMListEntry->GetLocalRect()

Get the size of a list entry

```
void ASAPI (*GetLocalRect)(ADMListEntryRef inEntry, ASRect* localRect);
```

Description

The **GetLocalRect()** function gets the size of **inEntry** in (0,0)-based dimensions.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

outLocalRect	The size of inEntry in (0,0)-based dimensions. The bottom and right members of the ASRect structure are the entry's size. Type: ASRect (see ASTypes.h)
---------------------	---

Returns

None.

sADMListEntry->GetPicture()

Gets a picture based on an icon

```
ADMLIconRef ASAPI (*GetPicture)(ADMListEntryRef inEntry);
```

Description

The **GetPicture()** function returns the picture used to draw **inEntry**.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

An ADM icon.

See also

[sADMListEntry->SetPicture\(\)](#)
[sADMListEntry->GetPictureID\(\)](#)
[sADMListEntry->GetSelectedPicture\(\)](#)
[sADMListEntry->GetSelectedPictureID\(\)](#)
[sADMListEntry->GetDisabledPicture\(\)](#)
[sADMListEntry->GetDisabledPictureID\(\)](#)

sADMListEntry->GetPictureID()

Get the list entry's picture ID

```
void ASAPI (*GetPictureID)(ADMListEntryRef inEntry, ASInt32*  
inPictureResID, const char** outPictureResName);
```

Description

This **GetPictureID()** function gets the resource ID of the picture used to draw an entry. If the item does not use a picture, **outPictureResID** returns 0.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

outPictureResID	Resource ID for the picture. If the entry does not have a picture, returns 0.
outPictureResName	Picture resource name.

Returns

None.

See also

[sADMListEntry->SetPictureID\(\)](#)
[sADMListEntry->GetPicture\(\)](#)
[sADMListEntry->GetSelectedPicture\(\)](#)
[sADMListEntry->GetSelectedPictureID\(\)](#)
[sADMListEntry->GetDisabledPicture\(\)](#)
[sADMListEntry->GetDisabledPictureID\(\)](#)

sADMListEntry->GetSelectedPicture()

Gets the picture currently selected in the list entry

```
ADMIconRef ASAPI (*GetSelectedPicture)(ADMListEntryRef
inEntry);
```

Description

The **GetSelectedPicture()** function gets the picture to be displayed when **inEntry** is selected.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

An ADM icon.

See also

[sADMListEntry->SetSelectedPicture\(\)](#)
[sADMListEntry->GetSelectedPictureID\(\)](#)
[sADMListEntry->GetPicture\(\)](#)
[sADMListEntry->GetPictureID\(\)](#)
[sADMListEntry->GetDisabledPicture\(\)](#)
[sADMListEntry->GetDisabledPictureID\(\)](#)

sADMListEntry->GetSelectedPictureID()

Get the list entry's selected picture ID

```
void ASAPI (*GetSelectedPictureID)(ADMListEntryRef inEntry,  
ASInt32* outPictureResID, const char** outPictureResName);
```

Description

This function returns the resource ID of the picture used to draw a list entry when it is selected. If the list entry does not have a selected picture, it will return 0.

The **GetSelectedPictureID()** function gets the resource ID of the picture used to draw **inEntry** when it is selected. If the list entry does not have a picture to display when it is selected, **outPictureResID** returns 0.

Parameters

inEntry	An ADM list entry.
outPictureResID	Resource ID for the picture. If the entry does not have a disabled picture, returns 0.
outPictureResName	Picture resource name.

Returns

None.

See also

[sADMListEntry->SetSelectedPictureID\(\)](#)
[sADMListEntry->GetSelectedPicture\(\)](#)
[sADMListEntry->GetPicture\(\)](#)
[sADMListEntry->GetPictureID\(\)](#)
[sADMListEntry->GetDisabledPicture\(\)](#)
[sADMListEntry->GetDisabledPictureID\(\)](#)

sADMListEntry->GetText()

Get the list entry's text

```
void ASAPI (*GetText)(ADMListEntryRef inEntry, char* outText,  
ASInt32 inMaxLength);
```

Description

The **GetText()** function retrieves a list entry's text and places it into the already allocated buffer pointed to by **outText**.

Parameters

inEntry	An ADM list entry.
outText	Buffer for inEntry 's text.
inMaxLength	Size of the buffer.

Returns

None.

See also

[sADMListEntry->SetText\(\)](#)

sADMListEntry->GetTextLength()

Get the length of the list entry's text

```
ASInt32 ASAPI (*GetTextLength)(ADMListEntryRef inEntry);
```

Description

The **GetTextLength()** function gets the number of characters in **inEntry**'s text.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

Number of characters in **inEntry**'s text.

sADMListEntry->GetUserData()

Get the user data of an ADM list entry

```
ADMUserData ASAPI (*GetUserData)(ADMListEntryRef inEntry);
```

Description

The **GetUserData()** function returns the 4-byte user value stored with **inEntry**. The meaning of the value is defined by **inEntry**'s creator. Commonly it is a pointer to a data structure—for instance, several values which are combined to make up the entry text. For some entries, it might be a simple 4-byte type, such as a long or a fixed number.

Each ADM list entry's user data is independent of the other list entries and its list item's data.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

The 4-byte user value stored with **inEntry**. Type: **ADMUserData** (see `ADMTypes.h`)

See also

[sADMListEntry->SetUserData\(\)](#)

sADMListEntry->GetVisualHierarchyDepth()

Get the visible depth of a hierarchy list entry

```
ASInt32 ASAPI (*GetVisualHierarchyDepth)(ADMListEntryRef
inEntry);
```

Description

The **GetVisualHierarchyDepth()** function gets the visible depth of **inEntry**. This is used to determine how much of the list is showing in the viewable portion of a dialog. For example, you can determine whether or not the parent list is hidden.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

Visible depth of **inEntry**.

See also

[sADMListEntry->GetHierarchyDepth\(\)](#)

sADMListEntry->HideEntryName()

Hides a hierarchy list entry's name

```
void ASAPI (*HideEntryName)(ADMListEntryRef inEntry, ASBoolean
inHideName);
```

Description

The **HideEntryName()** function hides the name of **inEntry**. This hides the parent list entry and forces the display of only the children.

This functionality is useful when you want to display a number of elements in a particular grouping order but not necessarily show the organizational grouping element. For example, the floating dialog palette shown below shows two types of

brush entries. Each type is grouped together (as shown by the icon indicators on the right hand side). In actuality each group has a parent hierarchy list that is hidden.



Parameters

inEntry	An ADM list entry.
inHideEntry	If true , the parent list entry is hidden and only the children are displayed; if false , the parent list entry is shown along with the children.

Returns

None.

sADMListEntry->Invalidate()

Invalidate the area of a list entry

```
void ASAPI (*Invalidate)(ADMListEntryRef inEntry);
```

Description

The **Invalidate()** function invalidates **inEntry**'s bounds within the dialog's window. This causes **inEntry** to be redrawn next time the screen is updated.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

None.

sADMListEntry->IsActive()

Determine whether an ADM list entry is active or inactive

```
ASBoolean ASAPI (*IsActive)(ADMListEntryRef inEntry);
```

Description

The **IsActive()** function determines whether **inEntry** is currently active. If so, it returns **true**; if not, it returns **false**.

To change its state, use the [sADMListEntry->Activate\(\)](#) function.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if **inEntry** is active; **false** otherwise.

See also

[sADMListEntry->Activate\(\)](#)

sADMListEntry->IsChecked()

Find out whether an ADM list entry is checked

```
ASBoolean ASAPI (*IsChecked)(ADMListEntryRef inEntry);
```

Description

The **IsChecked()** function determines whether **inEntry** is currently checked. If so it returns **true**; if not it returns **false**. To change its state, use the [sADMListEntry->Check\(\)](#) function.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if **inEntry** is checked; **false** otherwise.

See also[sADMListEntry->Check\(\)](#)**sADMListEntry->IsChildSelectable()**

Determines whether a list entry's child is selectable

```
ASBoolean ASAPI (*IsChildSelectable)(ADMListEntryRef inEntry);
```

Description

The **IsChildSelectable()** function determines whether **inEntry**'s child list entries are selectable. A use of a non-selectable child would be to display additional information about the child's parent entry.

NOTE: **inEntry** can have only one child list, which can itself hold multiple entries. This API returns whether or not they are selectable. The API can also be thought of as **AreChildrenSelectable**.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if **inEntry**'s child list entries are selectable; **false** otherwise.

sADMListEntry->IsEnabled()

Determines whether an ADM list entry is enabled

```
ASBoolean ASAPI (*IsEnabled)(ADMListEntryRef inEntry);
```

Description

The **IsEnabled()** function determines whether **inEntry** is currently enabled. If so, it returns **true**; if not, it returns **false**.

To change its state, use the [sADMListEntry->Enable\(\)](#) function. A disabled ADM list entry is dimmed and unusable.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if **inEntry** is enabled; **false** otherwise.

See also[sADMListEntry->Enable\(\)](#)

sADMListEntry->IsEntryNameHidden()

Determines whether a list entry's name is visible or hidden

```
ASBoolean ASAPI (*IsEntryNameHidden)(ADMListEntryRef inEntry);
```

Description

The **IsEntryNameHidden()** function determines whether the name of **inEntry** is visible or hidden.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if the name of **inEntry** is hidden; **false** otherwise.

See also

[sADMListEntry->HideEntryName\(\)](#)

sADMListEntry->IsHierarchyExpanded()

Determines whether a hierarchy list is expanded

```
ASBoolean ASAPI (*IsHierarchyExpanded)(ADMListEntryRef  
inEntry);
```

Description

The **IsHierarchyExpanded()** function determines whether a hierarchy list for **inEntry** is expanded.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if the hierarchy list for **inEntry** is expanded; **false** otherwise.

sADMListEntry->IsInBounds()

Determines whether a list entry is in bounds

```
ASBoolean ASAPI (*IsInBounds)(ADMListEntryRef inEntry);
```

Description

The **IsInBounds()** function determines whether **inEntry** is visible within the bounds of the list.

If **inEntry**'s bounds rect overlaps hierarchy list item's local rect, it is in bounds (or if there's no scrollbar). If the list entry is visible in the hierarchy list, it is in bounds. Content scrolled off the bottom or top is out of bounds.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if **inEntry** is visible within the bounds of the list; **false** otherwise.

sADMListEntry->IsSelected()

Determines whether an ADM list entry is selected

```
ASBoolean ASAPI (*IsSelected)(ADMListEntryRef inEntry);
```

Description

The **IsSelected()** function determines whether **inEntry** is currently selected. If so, it returns **true**; if not, it returns **false**. To change its state, use the [sADMListEntry->Select\(\)](#) function.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if **inEntry** selected; **false** otherwise.

See also

[sADMListEntry->IsChildSelectable\(\)](#)
[sADMListEntry->IsSelected\(\)](#)
[sADMListEntry->AreChildrenSelected\(\)](#)
[sADMListEntry->Select\(\)](#)

sADMListEntry->IsSeparator()

Determine whether an ADM list entry is a separator line

```
ASBoolean ASAPI (*IsSeparator)(ADMListEntryRef inEntry);
```

Description

The **IsSeparator()** function determines whether **inEntry** is a separator. If so, it returns **true**; if not, it returns **false**.

To change its state, use the [sADMListEntry->MakeSeparator\(\)](#) function.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

true if **inEntry** is a separator; **false** otherwise.

See also

[sADMListEntry->MakeSeparator\(\)](#)
[ADMListEntry->SetDividingLineColor\(\)](#)

sADMListEntry->LocalToScreenPoint()

Convert a list entry point to coordinates in its dialog

```
void ASAPI (*LocalToScreenPoint)(ADMListEntryRef inEntry,
ASPoint* ioPoint);
```

Description

The **LocalToScreenPoint ()** function converts a point in **inEntry** to a point in the coordinate space of its parent dialog.

Parameters

inEntry	An ADM list entry.
ioPoint	A point in inEntry . Type: ASPoint (see ASTypes.h)

Returns

None.

See also

[sADMListEntry->LocalToScreenRect\(\)](#)
[sADMListEntry->ScreenToLocalPoint\(\)](#)

sADMListEntry->LocalToScreenRect()

Convert a list entry's bounds rectangle to dialog coordinates

```
void ASAPI (*LocalToScreenRect)(ADMListEntryRef inEntry,
ASRect* inRect);
```

Description

The **LocalToScreenRect ()** function converts a rectangle in **inEntry**'s coordinates to a rectangle in its parent dialog's coordinate space.

Parameters

inEntry	An ADM list entry.
inRect	A point in inEntry . Type: ASRect (see ASTypes.h)

Returns

None.

See also

[sADMListEntry->LocalToScreenPoint\(\)](#)
[sADMListEntry->ScreenToLocalRect\(\)](#)

sADMListEntry->MakeInBounds()

Force a list entry within set bounds

```
void ASAPI (*MakeInBounds)(ADMListEntryRef inEntry);
```

Description

The **MakeInBounds()** function forces **inEntry** to be visible within a set boundary.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

None.

sADMListEntry->MakeSeparator()

Make an ADM list entry into a separator line

```
void ASAPI (*MakeSeparator)(ADMListEntryRef inEntry, ASBoolean
inSeparator);
```

Description

The **MakeSeparator()** function makes **inEntry** into a separator. Menu lists can have separators that are used to divide their entries into categories.

NOTE: This state is valid for list box items, though unused. If you are implementing a custom drawer for a list's entries, you could use this value.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

None.

See also

[sADMListEntry->IsSeparator\(\)](#)
[ADMListEntry->SetDividingLineColor\(\)](#)

sADMListEntry->ScreenToLocalPoint()

Convert a dialog point to list entry coordinates

```
void ASAPI (*ScreenToLocalPoint)(ADMListEntryRef inEntry,
ASPoint* ioPoint);
```

Description

The **ScreenToLocalPoint()** function converts a point in dialog coordinates to a point relative to **inEntry**.

Parameters

inEntry	An ADM list entry.
ioPoint	A point in inEntry . Type: ASPoint (see ASTypes.h)

Returns

None.

See also

[sADMListEntry->LocalToScreenPoint\(\)](#)
[sADMListEntry->ScreenToLocalRect\(\)](#)

sADMListEntry->ScreenToLocalRect()

Convert a dialog's bounds rectangle to list entry coordinates

```
void ASAPI (*ScreenToLocalRect)(ADMListEntryRef inEntry,
ASRect* inRect);
```

Description

The **ScreenToLocalRect()** function converts a rectangle in dialog coordinates to a rectangle in the coordinate space of **inEntry**.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

inRect	A rect in inEntry . Type: ASRect (see ASTypes.h)
---------------	--

Returns

None.

See also

[sADMListEntry->LocalToScreenRect\(\)](#)
[sADMListEntry->ScreenToLocalPoint\(\)](#)

sADMListEntry->Select()

Select or deselect a list entry

```
void ASAPI (*Select)(ADMListEntryRef inEntry, ASBoolean
inSelect);
```

Description

The **Select()** function selects or deselects **inEntry**.

In the case of a single selection list, other list entries are deselected automatically.

NOTE: For menu lists, a list entry's active and selection state are the same.

Parameters

inEntry	An ADM list entry.
inSelect	If true , inEntry is selected; if false , inEntry is deselected.

Returns

None.

See also

[sADMListEntry->IsChildSelectable\(\)](#)
[sADMListEntry->IsSelected\(\)](#)
[sADMListEntry->AreChildrenSelected\(\)](#)

sADMListEntry->SendNotify()

Send a notification to a list entry

```
void ASAPI (*SendNotify)(ADMListEntryRef inEntry, const char*
inNotifierType);
```

Description

The **SendNotify()** function sends notification of the type **inNotifierType** to **inEntry**. The main notifier for ADM list entries is:

```
#define kADMUserChangedNotifier          "ADM User Changed Notifier"
```

You can also define other user notifier types.

Parameters

inEntry	An ADM list entry.
inNotifierType	Notifier type. The only standard notifier for ADM list entries is: <pre>#define kADMUserChangedNotifier "ADM User Changed Notifier"</pre>

Returns

None.

ADMListEntry->SetBackgroundColor()

Set the list entry's background color

```
void ASAPI (*SetBackgroundColor)(ADMListEntryRef inEntry,  
    ADMColor inColor);
```

Description

The **SetBackgroundColor()** function sets **inEntry**'s background color.

Parameters

inEntry	An ADM list entry.
inColor	The color for the background. Type: ADMColor (see ADMTypes.h)

Returns

None.

sADMListEntry->SetDisabledPicture()

Set the list entry's disabled picture

```
void ASAPI (*SetDisabledPicture)(ADMListEntryRef inEntry,  
    ADMIconRef inPicture);
```

Description

The **SetDisabledPicture()** function sets the picture that is used when **inEntry** is disabled.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

inPicture	An ADM icon.
------------------	--------------

Returns

None.

See also

[sADMListEntry->GetDisabledPicture\(\)](#)
[sADMListEntry->SetDisabledPictureID\(\)](#)
[sADMListEntry->SetPicture\(\)](#)
[sADMListEntry->SetPictureID\(\)](#)
[sADMListEntry->SetSelectedPicture\(\)](#)
[sADMListEntry->SetSelectedPictureID\(\)](#)

sADMListEntry->SetDisabledPictureID()

Set the list entry's disabled picture ID

```
void ASAPI (*SetDisabledPictureID)(ADMListEntryRef inEntry,
ASInt32 inPictureResID, const char* inPictureResName);
```

Description

The **SetDisabledPictureID()** function sets the ID of the picture to be displayed for **inEntry** when it is disabled. **inPictureResID** is the ID of a platform picture or icon resource.

If the list entry does not have a disabled picture, ADM will gray the default picture when the list entry is disabled.

Parameters

inEntry	An ADM entry.
inPictureResID	Resource ID for the picture.
inPictureResName	Picture resource name.

Returns

None.

See also

[sADMListEntry->GetDisabledPictureID\(\)](#)
[sADMListEntry->SetDisabledPicture\(\)](#)
[sADMListEntry->SetPicture\(\)](#)
[sADMListEntry->SetPictureID\(\)](#)
[sADMListEntry->SetSelectedPicture\(\)](#)
[sADMListEntry->SetSelectedPictureID\(\)](#)

ADMListEntry->SetDividingLineColor() Set the list entry's dividing line color

```
void ASAPI (*SetDividingLineColor)(ADMListEntryRef inEntry,  
    ADMColor inColor);
```

Description

The **SetDividingLineColor()** function sets **inEntry**'s dividing line color. Default behavior is restored by specifying **kADMDummyColor**.

NOTE: A separator is the line that goes between menu items and takes up a place in the menu list. The dividing line is just the line drawn to separate list entries.

Parameters

inEntry	An ADM list entry.
inColor	The color for the background. Type: ADMColor (see ADMTypes.h)

Returns

None.

See also

[sADMListEntry->MakeSeparator\(\)](#)
[sADMListEntry->IsSeparator\(\)](#)

ADMListEntry->SetEntryItem() Set the list entry's item

```
void ASAPI (*SetEntryItem)(ADMListEntryRef inEntry, ADMItemRef  
    inItem);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **SetEntryItem()** function sets **inEntry**'s item.

Parameters

inEntry	An ADM list entry.
inItem	The ADM item to associate with inEntry .

Returns

None.

sAMListEntry->SetEntryTextRect()

Set the list entry's edit field

```
void ASAPI (*SetEntryTextRect)(ADMListEntryRef inEntry,
ASRect* inRect);
```

Description

The **SetEntryTextRect()** function sets the editable text field for **inEntry**. Used for in-place editing of text. The text rect is used to display the edit field.

Parameters

inEntry	An ADM list entry.
inRect	A rect in inEntry . Type: ASRect (see ASTypes.h)

Returns

None.

sAMListEntry->SetFont()

Set the list entry's font

```
void ASAPI (*SetFont)(ADMListEntryRef inEntry, ADMFont
inFont);
```

Description

The **SetFont()** function sets **inEntry**'s font.

Parameters

inEntry	An ADM list entry.
inFont	inEntry 's font style. Type: ADMFont (see ADMTTypes.h)

Returns

None.

sADMListEntry->SetID()

Set the ID of an ADM list entry

```
void ASAPI (*SetID)(ADMListEntryRef inEntry, ASInt32
inEntryID);
```

Description

The **SetID()** function sets the ID of **inEntry**. If you create the list entry, it initially has an ID of 0. You should set it within the list entry init function.

Parameters

inEntry	An ADM list entry.
inEntryID	ID for inEntry .

Returns

None.

See also

[sADMListEntry->GetID\(\)](#)

sADMListEntry->SetPicture()

Sets a picture based on an icon

```
void WINAPI (*SetPicture)(ADMListEntryRef inEntry, ADMLIconRef
inPicture);
```

Description

The **SetPicture()** function sets a picture based on an icon.

Parameters

inEntry	An ADM list entry.
inPicture	An ADM icon.

Returns

None.

See also

[sADMListEntry->GetPicture\(\)](#)
[sADMListEntry->SetPictureID\(\)](#)
[sADMListEntry->SetDisabledPicture\(\)](#)
[sADMListEntry->SetDisabledPictureID\(\)](#)
[sADMListEntry->SetSelectedPicture\(\)](#)
[sADMListEntry->SetSelectedPictureID\(\)](#)

sADMListEntry->SetPictureID()

Set the list entry's picture ID

```
void WINAPI (*SetPictureID)(ADMListEntryRef inEntry, ASInt32
inPictureResID, const char* inPictureResName);
```

Description

The **SetPictureID()** sets the ID for the picture to be displayed for **inEntry**. The **inPictureResID** is the ID of a platform picture or icon resource.

Parameters

inEntry	An ADM list entry.
inPictureResID	Resource ID for the picture.
inPictureResName	Picture resource name.

Returns

None.

See also

[sADMListEntry->GetPictureID\(\)](#)
[sADMListEntry->SetPicture\(\)](#)
[sADMListEntry->SetDisabledPicture\(\)](#)
[sADMListEntry->SetDisabledPictureID\(\)](#)
[sADMListEntry->SetSelectedPicture\(\)](#)
[sADMListEntry->SetSelectedPictureID\(\)](#)

sADMListEntry->SetSelectedPicture()

Set the list entry's selected picture

```
void ASAPI (*SetSelectedPicture)(ADMListEntryRef inEntry,
ADMIconRef inPicture);
```

Description

The **SetSelectedPicture()** function sets the picture to be displayed when **inEntry** is selected. If the list entry does not have a picture to use when **inEntry** is selected, ADM inverts the default picture to show that it is selected.

Parameters

inEntry	An ADM list entry.
inPicture	An ADM icon.

Returns

None.

See also

[sADMListEntry->GetSelectedPicture\(\)](#)
[sADMListEntry->SetSelectedPictureID\(\)](#)

[sADMListEntry->SetPicture\(\)](#)
[sADMListEntry->SetPictureID\(\)](#)
[sADMListEntry->SetDisabledPicture\(\)](#)
[sADMListEntry->SetDisabledPictureID\(\)](#)

sADMListEntry->SetSelectedPictureID()

Set the list entry's selected picture ID

```
void WINAPI (*SetSelectedPictureID)(ADMListEntryRef inEntry,
ASInt32 inPictureResID, const char* inPictureResName);
```

Description

The **SetSelectedPictureID()** function sets the picture ID for the picture to be displayed for **inEntry** when is selected. The **inPictureResID** is the ID of a platform picture or icon resource. If the entry does not have a picture to use when **inEntry** is selected, ADM will invert the default picture to show that it is selected.

Parameters

inEntry	An ADM list entry.
inPictureResID	Resource ID for the picture.
inPictureResName	Picture resource name.

Returns

None.

See also

[sADMListEntry->GetSelectedPictureID\(\)](#)
[sADMListEntry->SetSelectedPicture\(\)](#)
[sADMListEntry->SetPicture\(\)](#)
[sADMListEntry->SetPictureID\(\)](#)
[sADMListEntry->SetDisabledPicture\(\)](#)
[sADMListEntry->SetDisabledPictureID\(\)](#)

sADMListEntry->SetText()

Set the list entry's title

```
void WINAPI (*SetText)(ADMListEntryRef inEntry, const char*
inText);
```

Description

The **SetText()** function sets **inEntry**'s text to the indicated C string.

Parameters

inEntry	An ADM list entry.
inText	Text for inEntry .

Returns

None.

See also

[sADMListEntry->GetText\(\)](#)

sADMListEntry->SetTextColor()

Set the list entry's text color

```
void ASAPI (*SetTextColor)(ADMListEntryRef inEntry, ADMColor
inColor);
```

Description

The **SetTextColor()** function sets **inEntry**'s text color.

Parameters

inEntry	An ADM list entry.
inColor	Text color for inEntry . Type: ADMColor (see ADMTypes.h)

Returns

None.

sADMListEntry->SetUserData()

Set the user data of an ADM list entry

```
void ASAPI (*SetUserData)(ADMListEntryRef inEntry, ADMUserData
inData);
```

Description

The **SetUserData()** function sets the 4-byte user value stored with **inEntry**. Each list entry has its own user data. If you want to store user data for all list entries, use the list's ADM item's user data.

To get the item's user data, get the entry's hierarchical list and then get the hierarchical list's item reference. With this you can get the user data directly.

Parameters

inEntry	An ADM list entry.
inUserData	The 4-byte user value stored with inEntry . Type: ADMUserData (see <code>ADMTypes.h</code>)

Returns

None.

sADMListEntry->Update()

Force an update of a list entry

```
void ASAPI (*Update)(ADMListEntryRef inEntry);
```

Description

The **Update()** function invalidates **inEntry**'s bounds rectangle and immediately updates its contents. The redraw will occur if **inEntry**'s bounds rect is both visible and "dirty."

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

None

See also

[sADMListEntry->Invalidate\(\)](#)

ADM Help Support

ADM has built-in support for ASHelp, a WinHelp-type help system. ASHelp uses WinHelp file definitions in a cross-platform fashion. Every list item has a helpID and the system can operate in contextual fashion. For example, selecting **Command ?** in Macintosh or in **Alt + F1** in Windows lets you click an item and see that item's help resource. For plug-ins to support help files, there must be a Plugin Help location in the **PiPL** resource. The following three functions are used with ASHelp.

NOTE: These APIs are deprecated in ADM V2.8.

sADMListEntry->GetHelpID()

Get the help ID of a list entry

```
ASHelpID ASAPI (*GetHelpID)(ADMListEntryRef inEntry);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **GetHelpID()** function gets the help ID for **inEntry**.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

The help ID. Type: **ASHelpID** (See **ASHelp.h**)

See also

[sADMListEntry->SetHelpID\(\)](#)

sADMListEntry->Help()

Calls the help routine associated with a list entry

```
void ASAPI (*Help)(ADMListEntryRef inEntry);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **Help()** function calls the help for **inEntry**.

Parameters

inEntry	An ADM list entry.
----------------	--------------------

Returns

None.

sADMListEntry->SetHelpID()

Set the help ID of a list entry

```
void ASAPI (*SetHelpID)(ADMListEntryRef inEntry, ASHelpID  
inHelpID);
```

Description

NOTE: This API is deprecated in ADM V2.8.

The **SetHelpID()** function sets the help ID for **inEntry**. **inHelpID** is the resource ID for the ASHelp resource.

Parameters

inEntry	An ADM list entry.
inHelpID	The resource ID for the ASHelp resource. Type: ASHelpID (See <code>ASHelp.h</code>)

Returns

None.

See also

[sADMEEntry->GetHelpID\(\)](#)

17

The ADM Notifier Suite

About the ADM Notifier Suite

The ADM Notifier suite lets you access the high level events happening within your plugin/user interaction. For low-level events, use the ADM Tracker suite functions (see [Chapter 18, “The ADM Tracker Suite”](#)).

Accessing the Suite

The ADM Notifier suite is referred to as:

```
#define kADMNotifierSuite "ADM Notifier Suite"
```

with the version constant:

```
#define kADMNotifierSuiteVersion2 2
```

NOTE: Determine the suite version number you are using by examining the `ADMNotifier.h` header file.

The suite is acquired as follows:

```
ADMNotifierSuite *sADMNotifier;  
error = SSPBasic->AcquireSuite(kADMNotifierSuite,  
    kADMNotifierSuiteVersion2, &sADMNotifier);  
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

ADM Notifier Functions

ADM notifiers are callback functions assigned to ADM objects. They allow your plug-in to be notified that the user has interacted with an object. A notifier function is called when the user interaction is complete—for instance, when the mouse button is released. To specify an ADM notifier function to use with an ADM object, you use an assignment function:

```
void ASAPI (*SetNotifyProc)(ADMObjectRef inObject,  
    ADMObjectNotifyProc inNotifyProc);
```

For ADM Dialog objects and ADM Item objects, this assignment function is found in the object suite. Notifier functions for ADM Entry objects and ADM List Entry objects are assigned to the list or hierarchy list, respectively, that contains them. All entries or list entries in an ADM List object or ADM Hierarchy List object have the same notifier function.

All ADM notifier callbacks have the following signature:

```
typedef void ASAPI (*ADMOBJECTNotifyProc)(ADMOBJECTRef inObject,
                                           ADMNotifierRef inNotifier);
```

The ***object*** argument is a reference to the dialog, item, or entry that is to be notified that a user event has occurred. The ***inNotifier*** argument is a reference to the notification event and is used with the functions in this suite to obtain information about the event.

All ADM objects have a default notifier function which provides their normal notification behavior. For instance, the default notifier for a **ADMRadioButtonItem** is to set its selected state to **true** and the selected state of other buttons in its group to **false**. You should always call the default notifier function of an object to ensure that standard behaviors occur. To call the default notifier, you use a function of the object suite:

```
void ASAPI (*DefaultNotify)(ADMOBJECTRef inobject, ADMNotifyRef
                             inNotifier);
```

You pass the **DefaultNotify()** function the arguments that were passed to your notifier function, for instance:

```
void mySquareNotifyHandler(ADMItemRef item, ADMNotifierRef notifier) {
    sADMItem->DefaultNotify(item, notifier);
}
```

Using ADM Notifier Functions

The functions in the ADM Notifier suite require an **ADMNotifierRef**, which is basically an event context. One of the arguments passed to your notifier function is a notifier reference, and it is passed to each of the ADM notifier functions:

```
void myDialogNotifyHandler(ADMDialogRef inDialog, ADMNotifierRef
                             inNotifier) {
    sADMItem->DefaultNotify(inDialog, inNotifier);

    if ( sADMNotifier->IsNotifierType(inNotifier,
                                     kADMZoomHitNotifier) {
        // handle the window zoom...
    }
}
```

ADM Notifier Types

There are a number of types of ADM notifiers received at certain times. These notifiers are:

TABLE 17.1 ADM Notifier Types and Their Purposes

Notifier	Purpose
kADMUserChangedNotifier	The default notifier, received by all ADM objects. This type applies to all notification events that can't be classified explicitly as one of the precise types below.
kADMBoundsChangedNotifier	Received when an object is resized. This is received by both ADM items and dialogs. If a dialog receives this notification and resizes its items, the resized items would then receive this notifier.
kADMLEntryTextChangedNotifier	Received by ADM Entries when a list entry's text has changed by in-place editing. Note that when, for instance, an edit text item's text is changed, a kADMUserChangedNotifier notifier is received.
kADMCloseHitNotifier	This is received by ADM dialogs when a window's close box is hit. It is your responsibility to hide the window. ADM does not do so automatically.
kADMZoomHitNotifier	This is received by ADM dialogs when a window's zoom box is hit. It is your responsibility to change the window size. ADM has no means of do this automatically.
kADMIntermediateChangedNotifier	This is received by ADM items when a user is in the process of changing some input data via a slider, etc., but has not yet completed the task.

TABLE 17.1 *ADM Notifier Types and Their Purposes*

Notifier	Purpose
kADMCycleNotifier	This is received by ADM dialogs when a user is double clicking or triple clicking in the title bar of a tab palette.
kADMCollapseNotifier	This is received by ADM dialogs when the user is collapsing the palette via the tab.
kADMEExpandNotifier	This is received by ADM dialogs when the user is expanding the palette via the tab.
kADMGroupShowNotifier	This is received by an ADM item group when it is shown.
kADMGroupHideNotifier	This is received by an ADM item group when it is hidden.
kADMWindowDragMovedNotifier	This is received by ADM dialogs when the user moves the dialog by dragging it.
kADMContextMenuChangedNotifier	This is received by ADM dialogs when an edit operation has occurred via a clipboard operation.
kADMWindowShowNotifier	This is received by ADM dialogs when the user is in process of showing a window.
kADMWindowHideNotifier	This is received by ADM dialogs when the user is in the process of hiding a window.
kADMWindowActivateNotifier	This is received by ADM dialogs when a window is activated (focus moves into the dialog area).
kADMWindowDeactivateNotifier	This is received by ADM dialogs when a window is deactivated (focus moves to another screen area).

TABLE 17.1 ADM Notifier Types and Their Purposes

Notifier	Purpose
kADMPNumberOutOfBoundsNotifier	This is received by ADM dialogs when a user enters a value that is greater or smaller than the min/max values for the entry.
Text Item Notifiers:	
kADMPreClipboardCutNotifier	This is received by ADM text edit items when a user has issued a cut command, but the cut has not yet occurred.
kADMPostClipboardCutNotifier	This is received by ADM text edit items after the clipboard cut operation has occurred.
kADMPreClipboardCopyNotifier	This is received by ADM text edit items when a user has issued a copy command, but the copy has not yet occurred.
kADMPostClipboardCopyNotifier	This is received by ADM text edit items after the clipboard copy operation has occurred.
kADMPreClipboardPasteNotifier	This is received by ADM text items when a user has issued a paste command, but the paste has not yet occurred.
kADMPostClipboardPasteNotifier	This is received by ADM text edit items after the clipboard paste operation has occurred.
kADMPreClipboardClearNotifier	This is received by ADM text edit items when a user has issued a clear the clipboard command, but the clear has not yet occurred.
kADMPostClipboardClearNotifier	This is received by ADM text edit items after the clipboard clear operation has occurred.

TABLE 17.1 ADM Notifier Types and Their Purposes

Notifier	Purpose
kADMPreTextSelectionChangedNotifier	This is received by ADM text edit items when user has issued a text selection change command but the change has not yet occurred.
kADMTextSelectionChangedNotifier	This is received by ADM text edit items after the text selection change operation has occurred.
kADMPreClipboardRedoNotifier	This is received by ADM text edit items when the user has issued a redo command, but the redo has not yet occurred.
kADMPostClipboardRedoNotifier	This is received by ADM text edit items after a redo command has occurred.
kADMPreClipboardUndoNotifier	This is received by ADM text edit items when the user has issued an undo command, but the undo has not yet
kADMPostClipboardRedoNotifier	This is received by ADM text edit items after an undo command has occurred.

ADM items automatically handle certain behaviors internally and not through their notification function. These behaviors include setting text values or popup list selections. If you want ADM items to interact, you need to use a notifier. As a matter of practice, you should always call the [sADMItem->DefaultNotify\(\)](#) function within your custom function, even though in many cases there is no default notification. The default behaviors of item notifiers include:

TABLE 17.2 Default Notification Handling of ADM Objects

Item Type	Standard Notification Behavior
kADMFrameType	None.
kADMPicturePushButtonType	
kADMPictureStaticType	
ADMTextPushButtonType	
ADMUserType	
ADMTextStaticType	
kADMTextStaticMultilineType	
kADMSliderType	
kADMScrollbarType	
kADMTextCheckBoxType	
kADMListBoxType	
kADMSpinEditType	
kADMTextEditType	
kADMTextEditMultilineType	
kADMPictureCheckBoxType	
kADMPictureRadioButtonType	Sets the state of other radio buttons in the group.
kADMTextRadioButtonType	
kADMPopupListType	When the popup is used, notifies the item only if a list entry was selected.
kADMTextEditPopupType	
kADMSpinEditPopupType	
kADMTextEditScrollingPopupType	
kADMHierarchyListBoxType	
kADMPopupMenuType	
kADMScrollingPopupListType	
kADMSpinEditScrollingPopupType	
kADMResizeType	Sends bounds changed notification.

ADM Notifier Suite Functions

sADMNotifier->GetDialog()

Get the dialog of the notifier

```
ADMDialogRef ASAPI (*GetDialog)(ADMNotifierRef inNotifier);
```

Description

The **GetDialog()** function gets the dialog within which **inNotifier** event occurred.

Parameters

inNotifier	An ADM notifier.
-------------------	------------------

Returns

Dialog within which **inNotifier** event occurred.

sADMNotifier->GetItem()

Get the item of the notifier

```
ADMItemRef ASAPI (*GetItem)(ADMNotifierRef inNotifier);
```

Description

The **GetItem()** function gets the ADM item that triggered **inNotifier**.

Parameters

inNotifier	An ADM notifier.
-------------------	------------------

Returns

Item that triggered **inNotifier**.

sADMNotifier->GetNotifierType()

Get the type of a notifier

```
void ASAPI (*GetNotifierType)(ADMNotifierRef inNotifier, char* outNotifierType, ASUInt32 inMaxLength);
```

Description

The **GetNotifierType()** function returns the type of **inNotifier** as a C string. You can then do a string comparison to determine the type.

Parameters

inNotifier	An ADM notifier.
outNotifierType	Type of notifier.
inMaxLength	Maximum length for C string outNotifierType .

Returns

None.

See also

[sADMNotifier->IsNotifierType\(\)](#)

sADMNotifier->IsNotifierType()

Determine the type of the notifier

```
ASBoolean ASAPI (*IsNotifierType)(ADMNotifierRef inNotifier,  
const char* inNotifierType);
```

Description

The **IsNotifierType()** function determines the type of **inNotifier**. Pass in one of the notification constants and this function returns whether the notification is of that type.

Parameters

inNotifier	An ADM notifier.
inNotifierType	Notifier type as constant (see <code>ADMNotifier.h</code> for all possible notifier types listed as constant).

Returns

true if **inNotifier** is of type **inNotifierType**; **false** otherwise.

See also

[sADMNotifier->GetNotifierType\(\)](#)

sADMNotifier->SkipNextClipboardOperation()

Skip next clipboard operation

```
void ASAPI (*SkipNextClipboardOperation)(ADMNotifierRef  
inNotifier, ASBoolean inSkip);
```

Description

The **SkipNextClipboardOperation()** function skips the next clipboard operation. This function is only valid from within a clipboard operation—for example, to abort a cut or paste that is already in progress. After a pre-action notification, if a **SkipNextClipboardOperation()** is called, then the host will receive a post-action notification.

Parameters

inNotifier	An ADM notification.
inSkip	If true , next clipboard operation specified by inNotifier is skipped.

Returns

None.

18

The ADM Tracker Suite

About the ADM Tracker Suite

The ADM Tracker suite lets you access the low level events or actions happening within your plug-in/user interaction. For high level events, use the ADM Notifier suite functions. The ADM tracker stores all of the “tracked” state information in the TrackerRef and it is a snapshot of what activity was in progress at the time the tracker was activated. Any GetTrackerInfo function will obtain the state of the events at that point in time, not the current state.

Accessing the Suite

The ADM Tracker suite is referred to as:

```
#define kADMTrackerSuite "ADM Tracker Suite"
```

with the version constant:

```
#define kADMTrackerSuiteVersion2 2
```

NOTE: Determine the suite version number you are using by examining the ADMTracker.h header file.

The suite is acquired as follows:

```
ADMTrackerSuite *sADMTracker;  
error = sSPBasic->AcquireSuite(kADMTrackerSuite,  
    kADMTrackerSuiteVersion2, &sADMTracker);  
if (error) . . . //handle error
```

For SuitePea errors, see SPErrorCodes.h.

ADM Trackers

ADM trackers are routines that track low-level user interaction with dialogs and dialog items. When used in conjunction with action masks (see ADMTracker.h for a complete listing of all trackable modifier keys and mouse key actions), you can use trackers to keep aware of what the user is doing in interacting with your plug-in dialog.

For example, by setting up an ADM action mask with the various conditions you want to track and having your initialization routine set up a tracker procedure to call a notify procedure, your plug-in code can be alerted when the user causes an event you want to track.

The sequence is as follows:

1. Select the item to be “tracked” with [sADMDialog->GetItem\(\)](#).

2. Set up a notifier routine to call when your UI button is pressed with [sADMItem->SetTrackProc\(\)](#)
3. Tell ADM what conditions to check for with the [sADMItem->SetMask\(\)](#) function. When the specified condition is encountered, your Tracker proc should handle the appropriate response.

ADM Tracker Suite Functions

sADMTracker->Abort()

Stop the tracking procedure

```
void ASAPI (*Abort)(ADMTrackerRef inTracker);
```

Description

The **Abort()** function quits the tracking procedure.

Parameters

inTracker	An ADM tracker.
------------------	-----------------

Returns

None.

sADMTracker->GetAction()

Determine what action triggered tracking

```
ADMAction ASAPI (*GetAction)(ADMTrackerRef inTracker);
```

Description

The **GetAction()** function determines what event (mouse-up, mouse-down, key-down, etc.) triggered the tracker.

Parameters

inTracker	An ADM tracker.
------------------	-----------------

Returns

The event that occurred. Type: **ADMAction** (see `ADMTracker.h`)

See also

[sADMTracker->TestAction\(\)](#)

sADMTracker->GetCharacter()

Get character

```
ADMChar ASAPI (*GetCharacter)(ADMTrackerRef inTracker);
```

Description

The **GetCharacter()** function to get the ASCII value of a character input from the keyboard.

Parameters

inTracker	An ADM tracker.
------------------	-----------------

Returns

The ASCII character. Type: **ADMChar** (see `ADMTypes.h`)

sADMTracker->GetModifiers()

Determine which modifier keys are active

```
ADMModifiers ASAPI (*GetModifiers)(ADMTrackerRef inTracker);
```

Description

The **GetModifiers()** function obtains which, if any, modifier keys (**Command**, **Alt**, **Option**, etc.) are active. If **NULL** is passed for **inTracker**, the current modifiers are returned. All possible modifiers are listed in the `ADMTracker.h` header file.

Parameters

inTracker	An ADM tracker.
------------------	-----------------

Returns

Modifier key(s) that are pressed. Type: **ADMModifiers** (see `ADMTracker.h`)

See also

[sADMTracker->TestModifier\(\)](#)

sADMTracker->GetMouseState()

Get the mouse state

```
ADMMouseState ASAPI (*GetMouseState)(ADMTrackerRef inTracker);
```

Description

The **GetMouseState()** function obtains the current mouse state.

Parameters

inTracker	An ADM tracker.
------------------	-----------------

Returns

The current mouse state. The available states include (see `ADMTracker.h`):

```
typedef enum {
    kADMMouseNormal,
    kADMMouseCaptured,
    kADMMouseUncaptured
} ADMMouseState;
```

sADMTracker->GetPoint()

Get the point coordinates of the cursor

```
void ASAPI (*GetPoint)(ADMTrackerRef inTracker, ASPoint*
outPoint);
```

Description

The **GetPoint()** function gets the cursor position when **inTracker** was created. If **NULL** is passed for the tracker, the current mouse is point is returned.

Parameters

inTracker	An ADM tracker.
outPoint	The cursor position when inTracker was created. Type: ASPoint (see <code>ASTypes.h</code>)

Returns

None.

sADMTracker->GetTime()

Get the time at which an action occurred.

```
ADMTIME ASAPI (*GetTime)(ADMTrackerRef inTracker);
```

Description

The **GetTime()** function gets the time at which an action occurred.

Parameters

inTracker	An ADM tracker.
------------------	-----------------

Returns

The time at which an action occurred (in number of milliseconds since the system was last started). Type: **ADMTIME** (see `ADMTracker.h`)

sADMTracker->GetVirtualKey()

Obtain state of virtual key

```
ADMChar ASAPI (*GetVirtualKey)(ADMTrackerRef inTracker);
```

Description

The **GetVirtualKey()** function gets the current state of the virtual keys. Virtual keys are a cross platform mapping of special keyboard keys. The full listing of virtual keys is in the `ADMTracker.h` header file.

Parameters

inTracker	An ADM tracker.
------------------	-----------------

Returns

State of the virtual keys. Type: **ADMChar** (see `ADMTYPES.h`)

sADMTracker->ReleaseMouseCapture()

Release the current mouse state

```
void ASAPI (*ReleaseMouseCapture)(ADMTrackerRef inTracker);
```

Description

The **ReleaseMouseCapture()** function releases the current mouse state. This function is currently unimplemented.

Parameters

inTracker	An ADM tracker.
------------------	-----------------

Returns

None.

sADMTracker->TestAction()

Determine what action occurred

```
ASBoolean ASAPI (*TestAction)(ADMTrackerRef inTracker,  
ADMAction inAction);
```

Description

The **TestAction()** function is used to query what action triggered the tracking function. The full list of possible actions is in the `ADMTracker.h` header file.

Parameters

inTracker	An ADM tracker.
inAction	The event that occurred. Type: ADMAction (see <code>ADMTracker.h</code>). This parameter is returned by sADMTracker->GetAction() .

Returns

true if **inAction** is the action that triggered the tracking function; **false** otherwise.

See also

[sADMTracker->GetAction\(\)](#)

sADMTracker->TestModifier()

Determines if a particular modifier is active

```
ASBoolean ASAPI (*TestModifier)(ADMTrackerRef inTracker,
    ADMModifiers inModifier);
```

Description

The **TestModifier()** function determines which modifiers are activated. If **NULL** is passed for **inTracker**, the current modifier status is tested. The full list of modifiers is in the `ADMTracker.h` header file.

Parameters

inTracker	An ADM tracker.
inModifier	The modifier key(s) pressed. Type: ADMModifiers (see <code>ADMTracker.h</code>). This parameter is returned by sADMTracker->GetModifiers() .

Returns

true if **inModifier** is pressed; **false** otherwise.

See also

[sADMTracker->GetModifiers\(\)](#)

A

ADM Folders and Files

Files in SDK ADM Folders

The ADM files should be part of an accompanying SDK for an Adobe host application. The files provide the supporting files you will need to work with your host application.

The core ADM files are listed in [Table A.1](#).

TABLE A.1 *Core files*

File	Description
ADM.txt	Definitions of ADM item types and Mac and Window resource information. This is the same information listed in Table 1.4 and Table 1.5 .
ADMBasic.h	ADM Basic Suite functions
ADMDialog.h	ADM Dialog Suite functions
ADMDialogGroup.h	ADM DialogGroup Suite functions
ADMDrawer.h	ADM Drawer Suite functions
ADMEEntry.h	ADM Entry Suite functions
ADMHierarchyList.h	ADM HierarchyList Suite functions
ADMIcon.h	ADM Icon Suite functions
ADMImage.h	ADM Image Suite functions
ADMItem	ADM Item Suite functions
ADMList.h	ADM List Suite functions
ADMListEntry.h	ADM List Entry Suite functions
ADMNotifier.h	ADM Notifier Suite functions
ADMResource.h	ADM Resource ID lists
ADMTracker.h	ADM Tracker Suite functions
ADMTypes.h	ADM resource, units, font, color, etc. definitions

The ADM folder of an SDK also contains other folders and files of interest, as shown in [Table A.2](#).

TABLE A.2 *Other folders and files*

Folder	Description
IADM	C++ interfaces, see Using the C++ Interfaces .
Legacy	Maintenance files

B

ADM Glossary

Activate — When applied to edit text items, the cursor has been set somewhere in the text, either by the plug-in/application, or by the user; when applied to entries, the user has selected it; when applied to dialogs, it means that a floating dialog has focus. The **Activate(bool)** calls activate/deactivate the associated list item.

Action — A low-level system event such as a mouse-up, mouse-down, key-down, etc. The ADM tracker fields **ADMActions**. Actions can be filtered using an **ADMActionMask**.

AGM — Adobe Graphics Manager. Using **AGM**, you can write directly to a graphics port, bypassing ADM (not recommended). **AGM** is not “exposed” to 3rd party developers in all Adobe applications.

ASPoint — A point on the screen, defined by **x** and **y** coordinates. A point can appear within the coordinate system (0,0 in the upper left, with **y** values increasing “downward” and **x** values increasing to the right) of an item, a dialog, or the screen.

ASRect — A rectangle on the screen, defined by bottom, left, top, and right corners (each as an **ASPoint**). A “rect” can appear within the coordinate system (0,0 in the upper left, with **y** values increasing “downward” and **x** values increasing to the right) of an item, a dialog, or the screen.

Bounding rect — The full size of a “rect” (usually a dialog), including the border.

Callback — A user-supplied routine that is written and registered with ADM. Must follow the signature provided in the header files. Prototype signatures are always procs (end with a “proc”), but the user must define her own name.

Clipping — An operation performed on a display element to change (decrease) its size in some way, possibly changing its shape.

Dialog — Along with **Item**, the base object of ADM, usually a rectangle that appears on the screen, filled with **Items**; can be modal, non-modal, or pop-up.

Dialog group — A group of dialogs. A dock is a group of palettes or dialogs.

Dividing line — The line drawn to separate list entries. To be distinguished from **Separator**.

Dock — A group of palettes or dialogs.

Docked palette — A palette that is currently residing in its defined spot within a dock of palettes.

Enable — Can be selected by user; if disabled, the object is grayed out (for example, an entry may be greyed out). A disabled ADM dialog is dimmed and unusable.

Entry — An element of a list. Not to be confused with **List entry**, which is an element in a hierarchical list.

Floating palette — A dialog that is currently visible by itself on the screen (i.e., not docked and part of a group of palettes).

Flyout — A type of **Pop-up dialog** that appears when the user clicks on the popup button of another dialog.

Focus — Is ready to receive user input, or is simply not in the background (while other items or dialogs are).

Hierarchy list — A list with a sublist. The sublists may in turn have sublists. An element of hierarchy lists is called a **List entry**.

Item — Along with **Dialog**, the base object of ADM. An item can be a radio button, a text box with editable text, a list, etc.

Item group — An item group collects a number of items together that need to respond to calls as a group. For example, you might have five items that all need to be enabled or disabled simultaneously. Once those items belong to a group, you just need to enable/disable the group.

Known — An item is in a “known” state if it has a “good” or valid value.

Leaf entry — A child in a hierarchical list. Regular (non-hierarchical) lists do not have leaves. An entry that has no list attached is a leaf. If the entry has a list attached it is referred to as just an **Entry**.

List — A group of **Entries**. Not to be confused with a hierarchical list.

List entry — An element of a hierarchical list. Not to be confused with just an **Entry**, which is an element of a regular flat list.

Local rect — The area with the rect, usually a dialog, that does not include the border of the rect.

Modal dialog — A **Dialog** that only exists for the amount of time it is on-screen. Must be dismissed by the user before the application can resume interacting with the user. To be distinguished from a **Non-modal dialog** or **Modeless dialog**.

Modeless dialog — Same as a **Non-modal dialog**.

Non-Leaf entry — A list entry that is not a child.

Non-modal dialog — A **Dialog** that can exist indefinitely on-screen and must get focus before it can be used. **Docks**, **Floating palettes**, and **Dialog groups** are all examples of **Non-modal dialogs**.

Notification — A high-level event such as an undo or redo. Not to be confused with an **Action**.

Notifier — The ADM component that fields **Notifications**.

Pixel — The smallest addressable point in a display. Widths and heights are specified in pixels in ADM, as are **ASPoints**.

Pop-up dialog — A special type of dialog that is invoked by the user, usually via a mouse click. A **Flyout** is an example of a pop-up dialog. A traditional modal dialog can only be dismissed by either the **OK** or **Cancel** button. The **Pop-up modal dialog** is similar to a modal dialog in that it only stays up while the end-user is using it. However

the **Pop-up modal dialog** has no **OK/Cancel** button, but is dismissed when the end user makes a selection, hits the **Esc** button, etc.

Position code — Used with docked palettes to determine which palette is first, second, third, etc., plus where the tab is located (1st, 2nd, etc.).

Proc — A procedure; often used synonymously with “callback,” though a “proc” can be invoked even though a user hasn’t written his own callback and registered it with ADM. A common **Proc** is the Init proc (initialization callback). Other procs include Notifier procs, Tracker procs, and Draw procs.

SuitePea interface — Refers to the Plug-in Component Architecture (PICA), which is implemented as a series of suites. Pointers to the suites are often coded as **suiteP**, hence the term SuitePea, or, sometimes, SweetPea. PICA is described in the *Adobe PICA Programmer’s Guide and Reference* (see [Supporting Documents](#) in the [Preface](#)).

Select — Same as is **Activate** except that it invalidates the entry, causing a redraw.

Separator — The line that goes between menu items and takes up a place in the menu list. To be distinguish from a **Dividing line**.

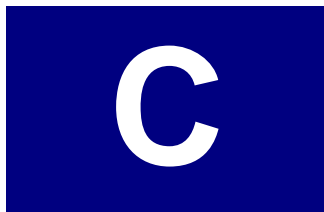
Tool palette — A floating palette with close boxes.

Tool tip — When the user moves his mouse over a GUI element (and does not click), a small indicator appears after some amount of (programmable) time that describes the GUI element.

Tracker — The ADM component that fields **ADMActions**.

Visible — Appears on-screen (Macintosh) or within the application **Window** (Windows).

Window — The basic element of display provided by an application.



Frequently Asked Questions (FAQ)

Frequently Asked Questions

The following is a compendium of frequently asked questions about ADM. They have been asked over the past several years. Some may only apply to older versions of ADM.

How does ADM handle platform native resources?

ADM uses platform native resources edited with the platform resource editor, so you have to maintain two sets of resources—one for the Mac, and one for Windows.

My hierarchical list entries are not showing. What do I need to do?

Call `sADMHierarchyList->SetEntryTextRect()`. The text rectangle for hierarchical lists defaults to an undefined rect.

I have an ADM frame item in my dialog with items inside of it. Those items do not seem to work. How do I make them work?

Move the frame item lower in the item list so it is the last item created. Don't just change the ID; open the `.rc` file as text and move the line defining the item.

I have an ADM numeric text item that has a min and max, but can also have no value to indicate that it is unused. When the user deletes the value, ADM puts a 0 back in the field. How can I get it to stop this behavior?

Use an ADM TextToFloat function (see [FloatToText and TextToFloat Functions](#) in Chapter 14, “The ADM Item Suite”). ADM has a default TextToFloat routine it uses. You can override it by using the `sADMItem->SetTextToFloatProc()` and you will have an opportunity to affect the float value used for the item.

If the TextToFloat proc returns `false`, the text is assumed to be invalid and a notification is presented to the user. If `true` is returned and the item is known (see `sADMItem->IsKnown()`), ADM checks it against the min and max values and acts accordingly.

If `true` is returned and your TextToFloat proc has marked the value as unknown, the text is used as is and no notification to the user is presented.

Can I use a Windows ActiveX control within an ADM item?

Yes, it has been done. This is not explicitly supported by Adobe Developer Support, so you're on your own in doing so. Look at the Q&A for MFC controls for possible help.

I want to fill a rect with color. How do I do this?

Use an ADM user item and override the Drawer proc. In the your Drawer proc, you can use `sADMDrawer->SetRGBColor()` and `sADMDrawer->FillRect()`.

What is an intermediate change notifier?

When a user action is completed for an item, such as a button press or tabbing out of a text item, a **kADMUserChangedNotification** is sent to the item. For sliders, where the user clicks on them and moves them, you might want some notification before the user lets up on the mouse. In this case, ADM sends **kADMIntermediateChangedNotifications** to the item. When the mouse is released, the user changed notification is sent.

I start doing something in my intermediate notification and it takes a long time and the control isn't very responsive anymore. What do I do?

You either scale back on what that “something” is, or you set a timer to see if the user has paused for a duration and do the update. ADM timers work especially well in this case.

How do timers work?

A timer takes an item or dialog reference, a duration in milliseconds, and two callback procs. The first callback proc is called if the duration expires; from the completion proc, you can return **true** and ADM repeats the timer. The second callback is called if the timer is aborted before the time duration. In this case, an abort mask is passed to the create timer call (see, for example, [sADMItem->CreateTimer\(\)](#)). If one of the actions in the mask occurs before the timer duration is finished, the abort proc is called. The action that caused the abort is passed to the callback.

How do I set and restore how palettes are tabbed together?

See the [Adobe Illustrator SDK](#) for a complete example. The gist of the process is the following:

See the [sADMDialogGroup->GetDialogGroupInfo\(\)](#) and [sADMDialogGroup->SetDialogGroupInfo\(\)](#) functions.

The “Get” function is used to retrieve the current palette positioning info so you can save it in your prefs. The “Set” function is used to restore your palette position. You need to have some default information for the first time the palette used.

Use the “Get” function in your dialog Destroy proc. This way, whenever your dialog is destroyed its position is saved; you don't have to worry about it being saved at shutdown or whenever else saving might need to occur. Similarly, use the “Set” function from within your dialog Init proc.

There are two values that are needed to save a tab palette position: the **groupName** and the **positionCode**. When you use the [sADMDialog->Create\(\)](#) function you specify a dialog name. This is meant to be a unique identifier and is used to determine a palette's position. The **groupName** returns this identifier for the top-left-most palette in a tab/dock group. All palettes in a group should have the same **groupName** identifier. The front tab is dealt with elsewhere, the **groupName** palette is an anchor.

The position code identifies where a palette is relative to the top left palette (**groupName**). There is info on this in the `ADMDialogGroup.h` header file, but, in a nutshell, it is a four-byte code. The low two bytes are 1-based indices specifying

where the palette is relative to the top left one. `0x00nn0101` is the top left, `0x00nn0201` is the second tab in the top tab group (top dock), `0x00nn0102` is the first tab in the second tab group (second dock), `0x00nn0202` is the second tab in the second dock, etc.

The 3rd byte is a group of bit flags. (There are masks in `ADMDialogGroup.h`.) Bit0 indicates whether the current dialog is the front-most tab; only one per tab group would have this set. Bit1 indicates whether the tab group is collapsed (1 is collapsed, 0 is expanded); all palettes in a tab group would probably have the same setting. Bit2 indicates whether the entire dock group (all tabs, all tab groups) is visible; all palettes in the collection would have the same setting (if not, the last one to set a position code would set the visibility for all).

In your prefs file, just write/read a C string and a `long`.

In Illustrator there is a file with all the palette name constants and dock codes. Since they are all relative to each other, they should be in the same place.

Do I always have to create all my palettes?

A palette wouldn't have to be created if it is not visible and stand-alone (not tabbed or docked with anything else—there is a function to determine this for a given position code). Other times a palette must be created because it is in a group or visible but stand-alone. The tendency is to create all palettes all the time.

Does the order in which I restore my palettes matter? Do I have to restore a group completely?

No. During restoration, the top-left dialog does not have to exist initially or at any time in the process. Other tab palettes can be missing and it won't affect the process. So, for instance, if you are operating in a plug-in palette world, palette restoration will work even if one or more of the plug-ins is removed.

Tell me where I'd get the proper ADM calls for tool bevel colors (3DShadow, 3D highlight, 3D Fill, etc)?

The ADM colors are in `ADMTypes.h`. Some of the colors may actually have the same RGB values, but by using the constants, the UI can change and will look correct.

For a given ADM drawing operation, you can set them with `sADMDrawer->SetADMColor()`. You can get the RGB representation of an ADM color with `sADMBasic->ADMColorToRGBColor()`.

I'm trying to create a resizable ADM dialog, but am unable to figure out how to get items inside of the dialog resizing correctly. The things that I've tried all end up in a stack overflow. How do I do this?

Attach a notifier proc to the resize item instead of the dialog. The notifier proc should look for bounds changed.

How do I create the pop up list for a palette—the one that is in the upper right of the window by the tabs?

Get the `kADMMenuItemID` item and from it get its list. When you populate the list with either a platform menu resource or by creating entries it will automatically be made visible.

How do I do a dialog tracker?

Most palettes/dialogs that need tracking need it at the item level. If they need to track the entire window, likely there is one or more items that cover that window. Simply make a tracker as big as the window—or for instance, as big as the window less scrollbars, and assign a tracker to it. Dialog trackers can be a bit tricky to use.

Dialog trackers are different from item trackers; dialog track procs should not expect that the actions sent to it are similar to those sent to an item's track proc. Dialog trackers can be used when the items in the dialog do not wish to handle an event. So the ADM tracker sends that event to the item's dialog to see if it can process the event. You can set up a dialog tracker to process such an event.

Example: A use of a dialog's track proc would be to trap keyboard events/modifiers for making shortcuts work.

I want to have a popup menu at the top of my preferences dialog so that I may have multiple sections. (Basically the same kind of functionality as the standard Adobe prefs dialog.) What is the solution with ADM?

Right now it is kind of awkward, but all of the items exist within one dialog and are hidden/shown or moved on/off-screen as needed. When ADM gets its own cross-platform resource, it will have panels that can be easily created and used.

We are laying out our ADM palettes and dialogs with PICTs for picture buttons. The backgrounds of the picture buttons sometimes differ from that of the dialog? How does ADM handle the system colors?

Do you mean, how do you get your pictures to blend in with whatever color scheme is currently being used? With pict/bmp resources, you can't; they always retain their palette. Currently the preferred way of doing picture buttons is using icons because of their masking feature, enabling backgrounds and other system colors to show through.

How can I tell when my palette window is visible?

There are two notifiers for this in `ADMNotifier.h`. Dialogs should now receive hide and show notification. The simplest case for this is when the close box is hit.

When a dialog is tabbed with others but not front-most, it is considered hidden. When it is brought to the front, the show notification is received. When another tab is selected, the hide notification is received.

All dialogs that are part of a dock group and the front-most tab of their tab group will receive hide/show notification based on a hide/show action to any other front tab in the dock group. This is because a hide/show on any visible palette affects any docked palettes similarly.

Using the tab key to hide/show all palettes triggers this notification for any visible dialog.

Note that the collapsed state of a tab does not affect its visible state and thus its corresponding menu state. Collapsing and expanding a tab will not trigger a hide/show notification.

Also note that when saving state information to the prefs file, the visible state of tabbed palettes should not be saved directly; the visibility of all palettes docked together is noted in the group position code.

I want to have a static text label select the text edit item next to it. How do I do this?

`ADMTexStatic` and `ADMPictureStatic` items auto activate the edit items they are next to. By default what they do is look at the next item to see if it is editable, if not it tries the previous item, and if that still is not editable it tries the one after the next item. (The last case is to handle labels on slider that have edit boxes.) This works in most cases.

With `kADMDisableAutoActiveTextStaticStyle` and `kADMDisableAutoActivatePictureStaticStyle` you can turn auto-activate off if this is not the behavior you want.

If it doesn't seem to work, check the ordering in the resource. Or disable the auto activate and on the `kADMUserChangedNotifier` activate the correct edit item.

For Windows static text items `kADMDisableAutoActivateTextStaticStyle` also disables converting ampersands to underscores for shortcuts in modal dialogs. Any Windows static text items that display user defined strings should turn this flag on in their dialog init code.

I want to bring up a system modal dialog and need to deactivate the ADM windows. How is this done?

To support deactivating palettes on the Mac when you bring up a system modal dialog, there is the `ActivateWindows()` call in `ADMHost.h`.

Can I mix platform functions such as the following with ADM suite functions?

```
ShowWindow(admGetWindowRef(dlgRef1), SW_SHOW);
```

No. There are often additional housekeeping things that ADM does that would likely get out of sync if the direct message is sent. Also, using the `sADMDialog->Show()` function keeps your code cross platform.

Why does a dialog have a name, an ID, and text?

The ID maps to the resource ID. The text is displayed in the tab or the title bar of the window. The name is an internal identifier and is not displayed to the user. It is used to restore dock groups and may be used for identifying resources in the future.

Calling the `sADMDialog->SetText()` for a dialog of type `kADMFloatingDialogStyle` doesn't make the title appear. It seems as if your WDEF doesn't display

them (I am working on a Mac). Is this by design, and is there any way around it?

[sADMDialog->SetText\(\)](#) is the correct way to do this. The text appears in the tab for tab palettes. Adobe doesn't put text in our title bars, so for non-tabbed windows, it wasn't hooked up. Bug or by design, it doesn't work in current/all versions of ADM.

I am experiencing problems getting ADM to load resources that are attached to the application and not a plug-in. Is there a way to set where ADM looks for resources? I don't have this problem from the Mac although it is the same code and the resources are in the application project and not in a plug-in.

When you add a dialog or palette (e.g. [sADMDialog->Create\(\)](#)) you specify a plug-in ref, even for the application-base ones. What did you specify here? You should create and save a single host plug-in ([SPAddHostPlugin\(\)](#)) and use this for all your application defined dialogs. When you add the host dialog, you can specify a file instance to use for resources. ADM will use this.

Can you send me a snippet/info on how to display a string after our name in the Splash Screen, so we can put up something like "Initializing QuickDraw 3D" or some such?

It goes something like the example below. Note this only works during the startup message. The string you pass should appear in the splash screen. Also, see `SPRuntime.h`.

```
SPERR error = kSPNoError;
SPHostProcs *gHostProcs;
error = sSPRuntime->GetRuntimeHostProcs(&gHostProcs);
if (!error)
{
    gHostProcs->startupNotify(kSetMessage,
        (void*)"Initializing QuickDraw 3d", gHostProcs->hostData);
}
```

I don't see anything in ADM to handle customized standard file dialogs (and their Win95 equivalent). Is this just not supported yet? Is there an intention to support it at some point? It'd be very useful...

ADM provides a standard file dialog, but the only customizable bit is the file filter proc that can be passed to it. There is currently no provision to add items, and since this is a system dialog there isn't a simple way to append an ADM item to it in a cross platform/product manner.

How do I draw into an ADM window at an arbitrary time?

You can invalidate the item and ADM will call your Drawer proc during the next update. If you need to do something like drag feedback where this is not possible, you can create a drawer for the window port. Remember to release it, though.

```
ASWindowRef windowRef = sADMDialog->GetWindowRef(sADMItem->GetDialog(item));
ASPortRef portRef = sADMDrawer->GetADMWindowPort(windowRef);
ADMDrawerRef drawer = sADMDrawer->Create(portRef, &boundsRect,
kADMDialogFont);
DrawFeedbackXOR(drawer, location);
sADMDrawer->Destroy((ADMDrawerRef)drawer);
sADMDrawer->ReleaseADMWindowPort(portRef);
```

I'm hard-coding a rectangle to tell me where to do a Photoshop::DisplayPixels(). Can this be expressed in a resource?

ADM provides a user item that you can use. It will do nothing if it isn't assigned drawer/tracker/notifier procs, so it can serve as a placeholder. The bounds can be retrieved with an ADM item call.

I have a question about list boxes and ADM. I'm trying to get a scrolling list box that allows items to be checked and unchecked and was wondering if there was an easy way to do this in ADM.

There isn't a built in way, but it isn't difficult to implement. Essentially, make two icons, checked and unchecked, put a track proc on the list. When the mouse goes down on the icon, use [sADMItem->SetPictureID\(\)](#) function to change the icon. Then call the default.

How do I tell if a text item is part of a composite item?

It would be nice if there was a `GetPartOf()` function, but there isn't. If you use composite items, you would need to keep track of the parent item and check for matches against its children using `GetChildItem()`.

Why does every edit item display with "pt" appended to the text?

This is just the default. You can turn off units for the individual item with the [sADMItem->SetUnits\(\)](#) function. You can set the defaults for all new text items with the host suite `SetADMDefaultTextInfo()` function.

I can't get an edit text to display centered. Why?

This would be on Windows, since the [sADMItem->SetJustify\(\)](#) function works on Mac. Windows does not support changes in justification to an single line edit item after it is created.

Is ADM thread safe?

ADM is as thread safe as the OS on which it is running. This is because ADM items are implemented using platform controls. So, for instance, on a Mac the answer is a definite "no" since most TB calls are not themselves thread safe. The only way around this is to eventually move away from platform controls and implement our own using

the ADM Drawer suite API. This has its own problems and is not on the list of immediate priorities.

Can ADM palettes/dialogs receive OS-level drag events?

Yes, it is possible. It is a host- or platform-provided service, meaning there is no support for drag-and-drop built into ADM. For instance, Illustrator implements drag and drop between the main document and the palettes and exports this as a SweetPea suite. This allows both the application and plug-ins to take advantage of it by acquiring the suite/functionality.

What essentially happens is the subscriber (app or plug-in with a palette) registers the data types it can accept and a callback for the drag action. The callback handles the feedback within the ADM dialog using standard ADM dialog, item, and drawer functions.

The call `sADMItem->SetCursorID()` is failing on the Mac but works on Windows.

The Mac version is missing the cursor resources. ADM recognizes 'CURS', not 'crsr'.

When you move any of the palettes in the Windows OS, the palette outline border is the same color as the main window. This results in not being able to see the palette outline when moving it. Expected results: use a different border shade/color from the main app window to make it visible. Default Windows color scheme was used. Anything we can do about it?

Not on the ADM side. This is a system setting. If you use the display control panel and turn off "show window contents when dragging", all apps have this behavior. Live dragging was disabled because docked palettes are actually several windows.

When I create a slider, it won't do anything.

Make sure you set the min and max. They are the same by default.

Is there a desirable width for floating palettes?

The default Adobe palette width is 206 pixels.

Is there any way to measure how many pixels wide a particular string displayed in a text item will be?

Currently only static text can be measured.

I've seen a reference to a `kADMNewDeleteStyle`, but it seems to be gone. Is there an equivalent? (I want a trash can at the bottom of my list, and I'll be adding a few custom icons, too.)

This style went away quite a while ago. Originally we thought it would be nice to have a way to create this type of list with one item since it was so common. This turned out to be a pain and now you just add the items yourself as pict push buttons. You can use the resize item for a relative position and height. The icons are included in ADM and you can set the picture ID to one of ADM's constants. See `ADMResource.h`.

I need to be able to have ADM not adjust the cursor over an item as we want to do our own cursor adjustment and not have ADM change it back.

There is a cursor constant that you can set and ADM won't bother changing the cursor at all: `kADMDHostControlsCursorID`.

Regarding this routine:

```
void ADMEdgeItem::InvalidateRect(IASRect &invalRect)
{
    if (GetWindowRef())
        ::InvalidateRect(GetWindowRef(), nil, false);
}
```

It is ignoring the passed rectangle. Has this been fixed already? If so, can I assume that the invalRect is passed in item-local-coordinates?

The implementation of this in some versions of ADM did invalidate the whole item. Where implemented correctly (current releases) the rect is local.

How is `kADMForegroundColor` used? Is there a foreground color that is stored even when you've switched to something like `kADMBackgroundColor`? If so, what is the proper way to save/restore the color state? Does calling `sADMDrawer->SetRGBColor()` implicitly switch you to `kADMForegroundColor` mode?

After Effects and other image processing programs will likely have a concept of a foreground color and a background color that affect certain operations. ADM's color scheme is very simple, being intended for UI work. The foreground color is currently always black (or gray for disabled items). That's it—no modes, just black. The background color is some variant of grey.

If you want to get and restore colors, you can use the ADM drawer color functions:

```
sADMDrawer->GetRGBColor()
sADMDrawer->SetRGBColor()
sADMDrawer->GetADMColor()
sADMDrawer->SetADMColor()
```

The `sADMDrawer->GetADMColor()` returns either the UI color constant in use or the RGB color, so it may be more convenient in some cases since you wouldn't need to do a look up yourself.

Should I save/restore both the RGB color and the ADM color? Or is the state reset to default at each draw event? Is my drawer private to only me?

A drawer is created for each drawing operation and whatever defaults exist are used. So in general, your drawer is reset and ready to use. There is one known bug with the clip rect for ADM entries, so we can't say that it is always fully reset. Assume it is and report strange behavior to Adobe Developer Support.

I want to handle the clipboard operation in a special way. How do I do this?

You need to override clipboard operations within an ADM dialog. To override the clipboard operation in a dialog, for all edit text items in the dialog attach a notifier that watches for the **kADMPreClipboard*** notifiers below. When you detect one, you can inspect the edit state and decide whether to allow it. For instance, you might look at the selection range.

If you decide you want to handle it in some fashion, you call the notifier suite function:

```
void ASAPI (*SkipNextClipboardOperation)(ADMNotifierRef notifier,
ASBoolean skip);
```

ADM will not do the default clipboard action. Presumably, you would do something appropriate within your notifier function. See the text item notifiers in [Table 17.1](#).

Do you have any examples I could start from, such as an ADM reference application?

There are many examples of the use of ADM from within plug-ins in the Adobe SDKs for Photoshop, Acrobat, Illustrator, and After Effects. See partners.adobe.com.

In response to certain user actions (for instance, a double click), I pop a modal dialog from within a tracker. I get some odd behaviors on the trackers I add to the new dialog.

The code used to display the dialog resides within the Tracker proc, resulting in nested trackers. ADM is not robust when it comes to handling nested trackers.

The thing to do is note that a double click occurred (and any other relevant info) and then call [sADMTracker->Abort\(\)](#) and return **true**. This allows ADM to clean up the current tracker and causes the notifier proc to be executed. In your notifier proc, check for a user changed notification and your double click flag and then pop the dialog. All should be well. If you can get the platform port for an AGM port, you can create an ADM drawer for it, though AGM support in ADM is deprecated in V2.8.

Is there anyway to set up an ADM drawer to draw into an AGM off-screen port? I have a UI item that I draw with AGM and then copy to the screen. It now has an icon as part of it. What I'm doing now is copying the AGM port, then using ADM to draw the icon. But when the item changes, it flickers. Can I use ADM to draw the icon into the AGM port? I also am using AGM to draw into an off-screen port for later use. I need to draw some text into this using the UI font. Drawing text with AGM is difficult and if I could use ADM to do it, it would be much easier.

AGM support in ADM is deprecated in V2.8. You can perform off-screen drawing in a Drawer proc with the ADM Image suite:

```
void ASAPI myDrawProc(ADMItemRef item, ADMDrawerRef inDrawer)
{
    imageRef = sADMImage->CreateOffscreen(width, height);
    if(imageRef != nil)
    {
        offscreenDrawer = sADMImage->BeginADMDrawer(imageRef);
```

```

        // draw stuff with offscreenDrawer

        sADMImage->EndADMDrawer(imageRef);
        topLeftPoint.h = 0;
        topLeftPoint.v = 0;
        sADMDrawer->DrawADMImage(inDrawer, imageRef, &topLeftPoint);
    }
}

```

How do I get ADM numeric text fields to not allow decimal numbers. When I set a precision of 0, I can still enter, for instance, “5.5” and it will truncate to “5.0”.

It’s a pretty simple change: for the precision, you should set `kADMIntegerNumeric` instead of 0.

I have written a plug-in and now I am implementing (functionality implementation). What I am trying to do is to have a group of buttons and each button responds in the same way. So I would place all buttons in a group box. Now when implementing in my plug-in the ideal way would be to get ONE notification when any of the buttons is being pushed that is on the Group. How would I implement this using ADM? Can I attach callback function directly to the group?

Use an ADM item group.

Can I have a notifier function set for an event as a Windows or a Mac event?

No. Notifiers are assignable to an ADM object, not an event. If you want to get low level events, use a tracker. ADM provides a cross platform event mechanism and there isn’t a way to get the system event.

Is there any compelling reason why some of my dialogs don't remember their last location on the screen, and instead always pop up in the same place? Is it an ADM bug or are these dialogs failing to do something they could do?

The “same place” should be centered on the screen, which is the default location if you don’t bother putting them somewhere else. The ones that remember their location are ADM dialogs where the plug-in does a `sADMDialog->GetBoundsRect()` in the Destroy proc and `sADMDialog->SetBoundsRect()` in the Init proc. Some of your dialogs may be system dialogs (e.g., print/file related ones) and there isn’t anything ADM can do for you there.

We want to have cmd/ctl do something like channel selection in Photoshop. ADM provides the Illustrator behavior (select the last active dialog). How can we change this?

ADM knows nothing about the Photoshop behavior, or what palettes exist in the host app and how to activate them. It can do nothing for this custom behavior.

You have two options:

1. Handle this on the host end by trapping this key code before calling `HandleADMMessage/HandleADMEvent()` to keep ADM from handling it. Then implement whatever behavior you want. Don't feed it to ADM.
2. Accept the default ADM behavior and let Photoshop be odd man out.

In my multi-line edit field double clicking a word then drag left/right/up/down does not highlight as you drag as it does in text edit programs. Single click and drag works normally though. I can't get kerning and tracking to preview in my ADM multiline edit text item. Hitting the Up or Down arrow does not take you to end-of-line or beginning-of-line.

ADM does not claim to offer a word processing text item. Standard platform text tricks are supported.

Severe flashing occurs with insertion cursor and text as you add or delete words. In some cases, the issue is so bad that the insertion cursor fails to correctly line up at the location of where you really are editing.

This is pretty much standard Windows behavior. Windows doesn't know how to set an insertion point in a single line edit text.

Sometimes when I click for an insertion point, it takes two or three seconds for the insertion cursor to catch up. This tends to happen after a lot of use of the type dialog or using large anti-aliased type.

You probably have the notifier doing some hefty processing when the mouse is clicked. There are no interruptible ADM notifiers.

The multi-line text edit field includes a scroll bar. Normally this is useful, except when I size the field to the full extent of the dialog. In this case, the down arrow on the scroll bar is hidden by the close box. I would normally fix this by making the scroll bar shorter than the text field itself.

All Adobe palettes are like this, presenting information above the resize box and using the area to the left for things like buttons. Your design should follow this layout and provide short cut buttons (or pictures or white space).

Resizing the field (by resizing the window) causes the scrollbar to reset to zero, and changes the selection.

Get a resize notifier and set the position wherever you think it should be.

There's no mechanism for appending text to the text edit item. I can get the text, append to that and re-set the text, but that's not an efficient solution for the purposes of an interactive console especially once the console contains a lot of text.

Use `sADMItem->GetChildItem()`. The child IDs are listed in `ADMItem.h`.

Is it possible to access the scroll bar for this multi-line text item and if so, how do I get a reference to it?

No solution.

sADMDialog->InvalidateRect() causes an update of the whole dialog. Is there a work around for this?

Invalidate the individual items and entries.

I can't get floating-point sliders to work with a range of [0,1]. I use the following for initializing:

```
sItem->SetMinFloatValue(i, min);
sItem->SetMaxFloatValue(i, max);
sItem->SetSmallIncrement(i, incr);
sItem->SetFloatValue(i, value);
```

where min, max, incr, and value are all floats. Using min = 0, max = 1, incr = .01, and value = .5, the slider thumb pops back and forth between 0 and 1 (the left and right endpoints). (Work around: I gave up and scaled the values into and out of the sliders.)

On a Mac (you don't mention the platform), the slider is a custom **CDEF**. This means that internally the values are integers. On Windows the slider code explicitly casts to an integer and then to a float, perhaps for compatibility with the Mac **CDEF**.

How would you recommend doing icon toggle buttons (i.e., an icon button that toggles on & off, rather than one that's off by default and only turns on when you hold the button down on it)?

There are picture check buttons, **kADMPictureCheckBoxType**. This should do what you want. There are also picture radio buttons for use in a group.

Is it possible to change the type of an item after a dialog is created?

You can't change dialog or item types after they have been created.

sADMDialog->Create() appears to be ignoring the "initially visible" bit in the Mac **DIALOG** resource. Is there a way to make a dialog initially invisible so I can set a bunch of parameters and move controls around without being visually disturbing?

In your Init proc you should do all your set up. The default **Show()** does not occur until after initialization, which should give you the effect you want. In your Init proc, you could do a **sADMDialog->Show()** with the boolean parameter set to **false**. The window will not appear until you do a **sADMDialog->Show()** with the boolean parameter set to **true**.

Is there a way to set the style of items in popup menus? I need to sometimes make items bold and/or italic. Do popup menus use a Mac menu on the Mac? If so, is there a way to get access to it?

No.

PICT items in Mac dialogs are stretched on display to fit the item rect, while in ADM, PICTs are apparently centered. Is there a way to get stretch-to-fit?

No. Adobe interfaces are pixel-perfect and so we would use the correct size picture.

How can I set a popup item list to use the small (palette) font?

You can't. Adobe doesn't mix fonts in dialogs (with the occasional exception of static text). This means you can't either.

The modal dialog I create doesn't prevent Photoshop from switching out. I'm using `sADMDialog->Create()` with `kADModalDialogStyle`, and then using `sADMDialog->DisplayAsModal()`, rather than using `sADMDialog->Modal()` directly. Is that OK?

In Adobe UI, you're supposed to be able to switch out of the application.

What are the resource parameters for creating a `kADMPictureCheckBoxType`?

Please see the platform-specific resource information in [Table 1.4](#) and [Table 1.5](#).

I'd like to change text fields back and forth between edit-text and static-text based on the settings of checkboxes. This works in the Mac dialog manager. I can't just disable an edit text item since text in disabled edit-text boxes disappears. That's annoying, as I'd like users to see the text, just not edit it.

The best you could do is make an edit and a static text. Instead of doing a set type, flip the one that is visible.

There's no call in the ADM List suite for selecting a single item. With Mac popups, this can be done easily with `SetControlValue`. With ADM popup lists, I have to go through the entries and disable everything first, then enable the newly selected item.

You could just get the active one and deactivate it, followed by activating the new active one.

I'd like a call in the ADM Entry suite to change the text style. I had to write a custom widget in FreeHand to do this; looks like I'll have to do the same for ADM.

You can override the drawer for the list and since you would use the ADM Drawer suite it will be cross-platform.

I want to set the text on my list item. Is the basic idea (using IADM):

```
IADMDEntry e = hl->InsertEntry(1);
IADMItem i = e->GetItem();
i->SetText("foo");
```

Support for IADM is deprecated in ADM V2.8. You wouldn't set the text on the item, but on the entry itself. You probably don't need the item reference/object.

```
IADMDEntry e = hl->InsertEntry(1);
e->SetText("foo");
```

What is the correct way to delete a dialog? In my dialog Notify proc, I look for the `kADMCloseHitNotifier` notifier. I then call `sADMDialog->DefaultNotify()` then `sADMDialog->Destroy()`, but ADM crashes.

The way this is implemented, you are essentially doing a delete **this** while **this** is still in use. The crash is expected. You probably want to note that dialog needs to be deleted and kill call `sADMDialog->Destroy()` during your idle proc.

I have some cases where the ADM CDEFs (for buttons, combo-boxes, etc.) aren't found. I think what's happening is that in the cases where plug-ins are not moved to the top of the resource chain, the CDEFs which are in app aren't found because it is at the top of the chain.

The expected resource chain order when ADM is used is app-ADM-plug-in. SP plug-ins usually get this for free. There are some host callback procs which, if specified, will allow you to set the resource chain for a SPPluginRef before ADM tries to do a resource access. See the *Adobe PICA Programmer's Guide and Reference* for details.

Am I right in assuming that if you cannot enter units, you can't enter math expressions?

No, units and math are orthogonal. You can do math without units, but you can't specify units in the operands.

I want to dismiss a modal dialog with a shortcut. How does `sADMDialog->EndModal()` work?

The `sADMDialog->EndModal()` function takes a parameter `ASBoolean inCancelling`. This should normally be `false` but can be `true` if you're implementing some other way of getting the **Cancel** functionality. When you call `sADMDialog->EndModal()` with `inCancelling` set to `true`, ADM doesn't verify the text in the current selection. If `inCancelling` is set to `false`, `sADMDialog->EndModal()` might return `false` (this is the place where you probably have to change your code) and if it does, your code should stop processing the notification and act as if nothing happened. What is happening in this case is that a numeric text item had the focus, the user typed in something illegal, ADM put up an alert, and so your code should abort the notification handling.

How does the “known” state for an ADM item work?

An item is in a “known” state if it has a “good” or valid value. For example, setting a checkbox item to `known(checkboxItem, false)` sets it to an intermediate state. The checkbox item then becomes “known” when it is checked by the user. The only way for an item to become “unknown” is by using the `known(someItem, false)` API. As another example, if you set a text item to unknown, it clears itself.

If you set the value of an item through the Set interfaces, it becomes known. If the user enters a value in a text item, it becomes known. If the value of a text item is unknown, it reverts to being empty in error conditions instead of reverting to its current value. If you have an item that you may set to unknown, you should check `sADMItem->IsKnown()` before getting its value.

Another example can be taken from the `kADMSpinEditPopupType` item. For the spin-edit as whole, “known” means the numerical value is known. Setting the parent item to unknown makes all of its child items unknown, but setting the parent item to known does not make its popup child known. (If the value is not on the menu, then the popup can be unknown even though the spin-edit is known.) If the popup is becoming known, it will do so by virtue of its value being set or by an entry being selected.

What is the return value for a track proc?

The return value from a tracker procedure can mean two different things. For keyboard actions it indicates whether the tracker “ate” the keystroke, or whether it should be propagated out to the surrounding environment. For other actions it indicates whether to call the notify procedure. Some cases have arisen where the latter meaning is required for keyboard actions.

My palette windows lose focus when the return key is pressed. This is interfering with my track procs ability to get key events.

The window with focus is the one that receives key events. Even though an ADM window is front-most, it doesn’t necessarily have focus. This is by design so that the document window has priority. The user can assign a palette window to have focus by doing a **cmd/opt** or **ctl/alt**-click on the window. It should then receive all key events.

What are the string lengths in use by ADM?

For the most part, and for ADM object text, the length is 256 (including the terminator). Tool tip text is allocated dynamically since tool tips may not be in use. Text identifiers (e.g., dialog names) are pooled strings. As such, they are subject to the SuitePea string pool limits. By default there isn’t a maximum length, but SuitePea allows the string pool to be substituted and so is subject to the host.

It appears to me that Windows ADM doesn't attempt to control the font used in popup menus. On my machine the popups appear to use MS Sans Serif instead of AdobeUI. The MS Sans Serif font doesn't have all the glyphs we have in the AdobeUI font so when we try to use fancy quotes in a popup menu they just show up as a vertical bar.

ADM uses standard platform controls where it can, and this is one of those cases.

My edit control is defaulting to a numeric style control and ADM tries to validate it and gives an error if there was ASCII text in the control. The resource definition looks like:

```
EDITTEXT ctUser1,102,35,70,12,ES_AUTOHSCROLL | ES_OEMCONVERT
```

How can I get it to be a normal text item by default?

Add the **ES_LEFT** style as:

```
EDITTEXT ctUser1,102,35,70,12,ES_AUTOHSCROLL | ES_OEMCONVERT | ES_LEFT
```

My list flickers when it draws. Is there a way to reduce this?

There is support for drawing entries in a listbox off-screen. Just add the style `kADMUseImageListBoxStyle`.

How does ADM handle the window's system colors?

ADM currently recolors bitmaps in `.icn` format, but not those in `.bmp`. The current recoloring scheme recolors all shades of gray in the original image. Colors are not recolored, and you can prevent recoloring by making pixels just slightly off-gray.

If the face color is dark, it ends up mapping a dark shade of gray; if it is light, it ends up mapping a light shade of gray. Everything else maps proportionally. So, for example, if the color scheme has the face at 25% intensity, the highlight at 10%, and the shadow at 50%, the grays map accordingly (50% gray becomes the shadow color when mapped). If the scheme is darker and has the face at 50%, the highlight at 25%, and the shadow at 75%, the grays still map accordingly (50% gray becomes the face color when mapped).

I just started using ADM to implement the tool palette and am having problems getting the pop-right flyout tools menu to track. I am creating a `kADMPopupDialogStyle` window and then adding a bunch of `kADMPictureRadioButtonType` items to it. I receive the mouse down and up messages to show and hide the submenu through the tracking proc fine but I don't get any mouse movement messages to track the menu. Should this work or do I have to track the menu via the Notify proc using the `kADMIntermediateChangedNotifier` message?

The ADM item in which the mouse down occurred captured the mouse, and all mouse actions go to that item until the mouse is released. So you should be getting `MouseMovedDown` actions for the item in which the button press occurred, which you have to translate into the space of the flyout in order to figure out what tool it is in.

The mouse-down item would “uncapture” the tracker, which then would allow events to be dispatched to the items in which they occur, and the mouse-down item would get an additional `UncapturedButtonUp` event when the mouse button is released so that it could know to pop the tool menu down.

Does ADM provide a way to get event information outside of a track proc?

For the ADM Tracker suite, if you pass `NULL` for `sADMTracker->GetModifiers()`, `sADMTracker->TestModifier()`, and `sADMTracker->GetPoint()`, they return the current state. (Normally they return the state of the keys when the event occurred in the tracker.) So for example if you need to check the state of the **Opt/Alt** key when your menu item is chosen, you would call:

```
if (sADMTracker->TestModifier(NULL, kADMModKeyDownModifier))
```

I have a dialog box that uses the `sADMItem->SetFont()` passing in the `kADMPaletteFont` on some items and normal on the other text items. Everything is fine on the Roman side—small text is small and normal text is normal. But when I run the US Photoshop on a Japanese OS on

Windows the text is all the same, normal size. My dialog box is very busy and I need the small text to be small. How do I get around this?

You can't. Non-roman fonts tend to be less usable at smaller sizes so though they exist, they are not supported.

I am trying to use `sADMHierarchyList->SetInitProc()`. Shouldn't the `OnListEntryInit()` function be called at the third line of code?

```
void ASAPI (*SetInitProc)(ADMHierarchyListRef list,
ADMListEntryInitProc initProc);

ADMHierarchyListRef listRef = sADMItem-
>GetHierarchyList(itemRef);

sADMHierarchyList->SetInitProc(listRef, OnListEntryInit);

ADMListEntryRef entry = sADMHierarchyList->InsertEntry(listRef,
index);
```

There's a bug in ADM that prevents Init procs from ever being called for hierarchy list entries.

How should I handle an error in my Init proc during modal dialog initialization?

Do not use `sADMDialog->EndModal()` in your initialization procedure. Instead, when an error occurs during initialization, the Init proc should return something other than `kNoErr`, and the dialog won't start up or show up.

What are the APIs that end with "W" in the name used for?

These are for internal use by Adobe. They are the "wide" functions used for Unicode support. Third-party developers should not use these APIs; you should use the corresponding APIs without the "W" instead.

On Windows, `Invalidate()` and `InvalidateRect()` do not erase the background when repainting.

Use the Windows invalidate calls with the `bErase` flag set to `true` in order to force the background to repaint.

Is it possible to construct dialogs without a platform resource?

Not in the current ADM. You can define a default empty dialog resource and use that over and over.

How do you specify `ADMHierarchyList` control in a `DIALOG` resource on Windows?

What you should do is to add a custom control item to the dialog layout when in the resource editor and then name the "class" of the custom control to be "ADM Hierarchy List Box Type" along with quotation marks.

I would like to know if it's possible to hide the scrollbar in an ADM ListBox. I have a listbox that will have a fixed number of items and does not need the disabled scrollbar.

Use the following code to hide the scrollbar.

```
scrollbarItem = sADMItem->  
GetChildItem(listItem,kADMListBoxScrollbarChildID);  
sADMItem->Show(scrollbarItem, false);
```




ADM Error Codes

The following error codes are returned wherever you see an **AS~~Err~~** returned by an API, or as the value of the **out~~Error~~** argument to the [sADMBasic->GetLastADMErr\(\)](#) call. In addition, you can return any of these errors from your initialization callbacks.

kNoErr	0
kOutOfMemoryErr	'!MEM'
kBadParameterErr	'PARM'
kNotImplementedErr	'!IMP'
kCantHappenErr	'CANT'
kADMCustomResourceError	'rErr'
kADMCustomResourceExistsError	'!Unq'
kADMStreamUnavailableError	'noSe'
kADMResourceNotFoundError	'r!fd'
kDialogResourceNotFoundError	'DLOG'
kDialogItemListResourceNotFoundError	'DITL'
kCouldntCreateItemError	'!itm'
kDockHostConflictError	'DOCK'
kTabGroupNotFoundError	'T!FD'
kAlreadyDockedError	'DCKD'
kASDataStreamErr	'DFER'
kASUnknownErr	'UNK '
kASBufferTooSmallErr	'BUFF'
kASMemoryErr	'MEM '

API Index

A

- Abort
 - sADMTTracker 538
- AbortTimer
 - sADMDialog 145
 - sADMEEntry 262
 - sADMIItem 373
 - sADMListEntry 486
- AboutBox
 - sADMBasic 117
- Activate
 - sADMDialog 146
 - sADMEEntry 262
 - sADMIItem 374
 - sADMListEntry 487
- AddItem
 - sADMIItem 374
- AdjustItemTabOrder
 - sADMDialog 146
- ADMCOLORToRGBColor
 - sADMBasic 131
- AreChildrenSelected
 - sADMListEntry 487
- AreToolTipsEnabled
 - sADMBasic 118
- AreToolTipsSticky
 - sADMBasic 119

B

- Beep
 - sADMBasic 119
- BeginADMDrawer
 - sADMIImage 357
- BeginAGMIImageAccess
 - sADMIImage 358
- BeginBaseAddressAccess
 - sADMIImage 358

C

- Check
 - sADMEEntry 263
 - sADMListEntry 488
- ChooseColor
 - sADMBasic 119
- Clear
 - sADMDrawer 221
- ClearRect
 - sADMDrawer 221
- CountDialogs
 - sADMDialogGroup 208
- Create
 - sADMDialog 147
 - sADMDrawer 221
 - sADMEEntry 263
 - sADMIIcon 350
 - sADMIImage 359
 - sADMIItem 375
 - sADMListEntry 488
- CreateBitmap
 - sADMIImage 360
- CreateChildList
 - sADMListEntry 489
- CreateFromImage
 - sADMIIcon 351
- CreateGroupInDialog
 - sADMDialog 150
- CreateItem
 - sADMDialog 151
- CreateMenu
 - sADMBasic 140
- CreateOffscreen
 - sADMIImage 360
- CreateTimer
 - sADMDialog 152
 - sADMEEntry 264
 - sADMIItem 376
 - sADMListEntry 489

D

- DefaultDraw
 - sADMDialog 153
 - sADMEEntry 265
 - sADMIItem 378
 - sADMListEntry 491
- DefaultFloatToText
 - sADMIItem 378
- DefaultNotify
 - sADMDialog 154
 - sADMEEntry 266
 - sADMIItem 379
 - sADMListEntry 491
- DefaultTextToFloat
 - sADMIItem 379
- DefaultTrack
 - sADMDialog 154
 - sADMEEntry 266
 - sADMIItem 380
 - sADMListEntry 492
- DeleteChildList
 - sADMListEntry 493
- DeselectAll
 - sADMHierarchyList 296
- Destroy
 - sADMDialog 155
 - sADMDrawer 222
 - sADMEEntry 267
 - sADMIIcon 351
 - sADMImage 361
 - sADMIItem 381
 - sADMListEntry 493
- DestroyItem
 - sADMDialog 156
- DestroyMenu
 - sADMBasic 140
- DisplayAsModal
 - sADMDialog 156
- DisplayAsPopupModal
 - sADMDialog 157
- DisplayMenu
 - sADMBasic 141
- DrawADMImage
 - sADMDrawer 223
- DrawADMImageCentered
 - sADMDrawer 223
- DrawAGMImage
 - sADMDrawer 224
- DrawDownArrow
 - sADMDrawer 225
- DrawIcon
 - sADMDrawer 225
- DrawIconCentered
 - sADMDrawer 226
- DrawLeftArrow
 - sADMDrawer 227
- DrawLine
 - sADMDrawer 227
- DrawOval
 - sADMDrawer 228
- DrawPolygon
 - sADMDrawer 228
- DrawRaisedRect
 - sADMDrawer 229
- DrawRecoloredIcon
 - sADMDrawer 230
- DrawRecoloredResPicture
 - sADMDrawer 231
- DrawRecoloredResPictureCentered
 - sADMDrawer 231, 232
- DrawRect
 - sADMDrawer 233
- DrawResPicture
 - sADMDrawer 234
- DrawResPictureCentered
 - sADMDrawer 235
- DrawRightArrow
 - sADMDrawer 235
- DrawSunkenRect
 - sADMDrawer 236
- DrawText
 - sADMDrawer 237
- DrawTextCentered
 - sADMDrawer 237
- DrawTextInaBox
 - sADMDrawer 238
- DrawTextLeft

- sADMDrawer 239
- DrawTextRight
 - sADMDrawer 239
- DrawUpArrow
 - sADMDrawer 240

E

- Enable
 - sADMDialog 158
 - sADMEEntry 268
 - sADMItem 382
 - sADMListEntry 494
- EnableChildSelection
 - sADMListEntry 494
- EnableTip
 - sADMItem 456
- EnableToolTips
 - sADMBasic 120
- EndADMDrawer
 - sADMImage 362
- EndAGMImageAccess
 - sADMImage 361
- EndBaseAddressAccess
 - sADMImage 362
- EndModal
 - sADMDialog 158
- ErrorAlert
 - sADMBasic 121
- ExpandHierarchy
 - sADMListEntry 495

F

- FillOval
 - sADMDrawer 241
- FillPolygon
 - sADMDrawer 241
- FillRect
 - sADMDrawer 242
- FindEntry
 - sADMHierarchyList 296
 - sADMList 463

G

- GetAction
 - sADMTracker 538
- GetActiveEntry
 - sADMHierarchyList 297
 - sADMList 463
- GetActiveLeafEntry
 - sADMHierarchyList 297
- GetADMColor
 - sADMDrawer 242
- GetADMWindowPort
 - sADMDrawer 243
- GetAGMPort
 - sADMDrawer 244
- GetAllowMath
 - sADMItem 382
- GetAllowUnits
 - sADMItem 383
- GetAppFPS
 - sADMBasic 131
- GetAppUnits
 - sADMBasic 132
- GetBackColor
 - sADMItem 383
- GetBestSize
 - sADMItem 384
- GetBitsPerPixel
 - sADMImage 363
- GetBooleanValue
 - sADMItem 384
- GetBoundsRect
 - sADMDialog 159
 - sADMDrawer 244
 - sADMEEntry 268
 - sADMItem 385
 - sADMListEntry 495
- GetByteWidth
 - sADMImage 363
- GetCancelItemID
 - sADMDialog 159
- GetCharacter
 - sADMTracker 539
- GetCheckGlyph

sADMEEntry 269
 GetChildItem
 sADMItem 385
 GetClipRect
 sADMDrawer 245
 GetCursorID
 sADMDialog 160
 sADMItem 387
 GetData
 sADMIcon 352
 GetDefaultIncrements
 sADMBasic 132
 GetDefaultItemID
 sADMDialog 160
 GetDestroyProc
 sADMDialog 161
 sADMHierarchyList 298
 sADMItem 387
 sADMList 464
 GetDialog
 sADMItem 388
 sADMNotifier 533
 GetDialogGroupInfo
 sADMDialogGroup 208
 GetDialogName
 sADMDialog 161
 sADMDialogGroup 209
 GetDialogStyle
 sADMDialog 162
 GetDisabledPicture
 sADMEEntry 269
 sADMItem 388
 sADMListEntry 496
 GetDisabledPictureID
 sADMEEntry 270
 sADMItem 389
 sADMListEntry 497
 GetDivided
 sADMHierarchyList 298
 GetDrawMode
 sADMDrawer 245
 GetDrawProc
 sADMDialog 162
 sADMHierarchyList 299
 sADMItem 389
 sADMList 465
 GetEntry
 sADMHierarchyList 299
 sADMList 465
 GetEntryHeight
 sADMHierarchyList 300
 sADMList 466
 GetEntryItem
 sADMListEntry 497
 GetEntryTextRect
 sADMHierarchyList 300
 sADMList 466
 GetEntryWidth
 sADMHierarchyList 301
 sADMList 467
 GetExpandArrowLocalRect
 sADMListEntry 498
 GetExpandedIndex
 sADMHierarchyList 301
 GetFixedValue
 sADMItem 390
 GetFlags
 sADMHierarchyList 302
 GetFloatToTextProc
 sADMItem 390
 GetFloatValue
 sADMItem 391
 GetFont
 sADMDialog 163
 sADMDrawer 246
 sADMItem 391
 GetFontInfo
 sADMDrawer 246
 GetForeColor
 sADMItem 392
 GetFromResource
 sADMIcon 352
 GetGlobalLeftMargin
 sADMHierarchyList 302
 GetHasRollOverProperty
 sADMItem 392
 GetHeight
 sADMIcon 353

- sADMIImage 363
- GetHelpID
 - sADMDialog 203
 - sADMEEntry 290
 - sADMItem 456
 - sADMListEntry 524
- GetHierarchyDepth
 - sADMListEntry 498
- GetHierarchyList
 - sADMItem 393
- GetHorizontalIncrement
 - sADMDialog 163
- GetID
 - sADMDialog 164
 - sADMEEntry 270
 - sADMItem 393
 - sADMListEntry 499
- GetIndentationWidth
 - sADMHierarchyList 303
- GetIndex
 - sADMEEntry 271
 - sADMListEntry 499
- GetIndexString
 - sADMBasic 116
- GetInitProc
 - sADMHierarchyList 303
 - sADMList 467
- GetIntValue
 - sADMItem 393
- GetItem
 - sADMDialog 165
 - sADMHierarchyList 304
 - sADMList 468
 - sADMListEntry 496, 500
 - sADMNotifier 534
- GetItemStyle
 - sADMItem 394
- GetItemType
 - sADMItem 394
- GetJustify
 - sADMItem 395
- GetLargeIncrement
 - sADMItem 395
- GetLastADMEError
 - sADMBasic 133
- GetLeafIndex
 - sADMHierarchyList 304
- GetList
 - sADMEEntry 271
 - sADMItem 396
 - sADMListEntry 500
- GetLocalLeftMargin
 - sADMHierarchyList 305
- GetLocalRect
 - sADMDialog 165
 - sADMEEntry 272
 - sADMHierarchyList 305
 - sADMItem 396
 - sADMListEntry 500
- GetMask
 - sADMDialog 166
 - sADMHierarchyList 306
 - sADMItem 397
 - sADMList 468
- GetMaxFixedValue
 - sADMItem 397
- GetMaxFloatValue
 - sADMItem 398
- GetMaxHeight
 - sADMDialog 166
- GetMaxIntValue
 - sADMItem 398
- GetMaxTextLength
 - sADMItem 398
- GetMaxWidth
 - sADMDialog 167
- GetMenuID
 - sADMHierarchyList 306
 - sADMList 469
- GetMinFixedValue
 - sADMItem 399
- GetMinFloatValue
 - sADMItem 399
- GetMinHeight
 - sADMDialog 167
- GetMinIntValue
 - sADMItem 400
- GetMinWidth

- sADMDialog 168
- GetModifiers
 - sADMTracker 539
- GetMouseState
 - sADMTracker 539
- GetNamedDialog
 - sADMDialogGroup 209
- GetNextItem
 - sADMDialog 168
- GetNonLeafEntryTextRect
 - sADMHierarchyList 307
- GetNonLeafEntryWidth
 - sADMHierarchyList 307
- GetNotifierData
 - sADMDialog 169
 - sADMHierarchyList 308
 - sADMItem 400
 - sADMList 469
- GetNotifierType
 - sADMNotifier 534
- GetNotifyProc
 - sADMDialog 169
 - sADMHierarchyList 308
 - sADMItem 401
 - sADMList 469
- GetNthDialog
 - sADMDialogGroup 210
- GetNumbersArePoints
 - sADMBasic 133
- GetOrigin
 - sADMDrawer 247
- GetPaletteLayoutBounds
 - sADMBasic 134
- GetParentEntry
 - sADMHierarchyList 309
- GetPicture
 - sADMEEntry 272
 - sADMItem 401
 - sADMListEntry 501
- GetPictureID
 - sADMEEntry 273
 - sADMItem 402
 - sADMListEntry 501
- GetPixel

- sADMImage 364
- GetPluginRef
 - sADMDialog 170
 - sADMItem 402
- GetPoint
 - sADMTracker 540
- GetPopupDialog
 - sADMItem 403
- GetPortRef
 - sADMDrawer 247
- GetPrecision
 - sADMItem 403
- GetPreviousItem
 - sADMDialog 170
- GetResPictureBounds
 - sADMDrawer 248
- GetRGBColor
 - sADMDrawer 248
- GetRolloverPicture
 - sADMItem 404
- GetRolloverPictureID
 - sADMItem 404
- GetScreenDimensions
 - sADMBasic 134
- GetSelectedPicture
 - sADMEEntry 273
 - sADMItem 405
 - sADMListEntry 502
- GetSelectedPictureID
 - sADMEEntry 274
 - sADMItem 406
 - sADMListEntry 503
- GetSelectionRange
 - sADMItem 406
- GetShowUnits
 - sADMItem 407
- GetSmallIncrement
 - sADMItem 407
- GetText
 - sADMDialog 171
 - sADMEEntry 274
 - sADMItem 408
 - sADMListEntry 503
- GetTextLength

- sADMDialog 172
- sADMEEntry 275
- sADMItem 408
- sADMListEntry 504
- GetTextRectHeight
 - sADMDrawer 249
- GetTextToFloatProc
 - sADMItem 409
- GetTextWidth
 - sADMDrawer 249
- GetThisFontInfo
 - sADMDrawer 250
- GetTime
 - sADMTracker 540
- GetTipString
 - sADMItem 457
- GetTipStringLength
 - sADMItem 457
- GetToolTipDelays
 - sADMBasic 121
- GetTrackProc
 - sADMDialog 172
 - sADMHierarchyList 309
 - sADMItem 409
 - sADMList 470
- GetType
 - sADMIcon 353
- GetUnits
 - sADMItem 410
- GetUpdateRect
 - sADMDrawer 250
- GetUserData
 - sADMDialog 173
 - sADMEEntry 275
 - sADMHierarchyList 310
 - sADMItem 410
 - sADMList 471
 - sADMListEntry 504
- GetVerticalIncrement
 - sADMDialog 173
- GetVirtualKey
 - sADMTracker 541
- GetVisualHierarchyDepth
 - sADMListEntry 505

- GetWantsFocus
 - sADMItem 411
- GetWidth
 - sADMIcon 354
 - sADMImage 364
- GetWindowRef
 - sADMDialog 174
 - sADMItem 411
- GetWorkspaceBounds
 - sADMBasic 135
- GlobalToLocalPoint
 - sADMHierarchyList 310
- GlobalToLocalRect
 - sADMHierarchyList 311

H

- Help
 - sADMDialog 204
 - sADMEEntry 291
 - sADMItem 458
 - sADMListEntry 524
- HideEntryName
 - sADMListEntry 505
- HideToolTip
 - sADMBasic 122
 - sADMItem 458

I

- IgnoreForceRoman
 - sADMItem 412
- IndexAllSelectedEntriesInHierarchy
 - sADMHierarchyList 311
- IndexEntry
 - sADMHierarchyList 312
 - sADMList 471
- IndexExpandedEntry
 - sADMHierarchyList 313
- IndexLeafEntry
 - sADMHierarchyList 313
- IndexSelectedEntry
 - sADMHierarchyList 314
 - sADMList 472

- IndexUnNestedSelectedEntriesInHierarchy
 - sADMHierarchyList 315
 - InsertEntry
 - sADMHierarchyList 315
 - sADMList 473
 - InsertGivenEntry
 - sADMHierarchyList 316
 - IntersectClipPolygon
 - sADMDrawer 251
 - IntersectClipRect
 - sADMDrawer 252
 - Invalidate
 - sADMDialog 174
 - sADMEEntry 276
 - sADMHierarchyList 316
 - sADMItem 412
 - sADMListEntry 506
 - InvalidateRect
 - sADMDialog 175
 - sADMEEntry 276
 - sADMItem 413
 - InvertRect
 - sADMDrawer 252
 - IsActive
 - sADMDialog 175
 - sADMEEntry 277
 - sADMItem 413
 - sADMListEntry 507
 - IsChecked
 - sADMEEntry 277
 - sADMListEntry 507
 - IsChildSelectable
 - sADMListEntry 508
 - IsCollapsed
 - sADMDialog 176
 - sADMDialogGroup 210
 - IsDialogContextHidden
 - sADMDialog 176
 - IsDockVisible
 - sADMDialogGroup 211
 - IsEnabled
 - sADMDialog 177
 - sADMEEntry 278
 - sADMItem 414
 - sADMListEntry 508
 - IsEntryNameHidden
 - sADMListEntry 509
 - IsForcedOnScreen
 - sADMDialog 177
 - IsFromResource
 - sADMIcon 354
 - IsFrontTab
 - sADMDialogGroup 211
 - IsHierarchyExpanded
 - sADMListEntry 509
 - IsInBounds
 - sADMEEntry 278
 - sADMListEntry 509
 - IsInRollOverState
 - sADMItem 414
 - IsKnown
 - sADMItem 415
 - IsNotifierType
 - sADMNotifier 535
 - IsSelected
 - sADMEEntry 279
 - sADMListEntry 510
 - IsSeparator
 - sADMEEntry 279
 - sADMListEntry 510
 - IsStandAlonePalette
 - sADMDialogGroup 212
 - IsTipEnabled
 - sADMItem 458
 - IsVisible
 - sADMDialog 178
 - sADMItem 415
- K**
- Known
 - sADMItem 416
- L**
- LightweightErrorAlert
 - sADMBasic 122
 - LoadToolTips

- sADMDialog 179
- LocalToGlobalPoint
 - sADMHierarchyList 317
- LocalToGlobalRect
 - sADMHierarchyList 318
- LocalToScreenPoint
 - sADMDialog 179
 - sADMEEntry 279
 - sADMHierarchyList 318
 - sADMLItem 416
 - sADMLListEntry 511
- LocalToScreenRect
 - sADMDialog 180
 - sADMEEntry 280
 - sADMLItem 417
 - sADMLListEntry 511
- LookUpZString
 - sADMBasic 135

M

- MakeInBounds
 - sADMEEntry 280
 - sADMLListEntry 512
- MakeSeparator
 - sADMEEntry 281
 - sADMLListEntry 512
- MessageAlert
 - sADMBasic 123
- Modal
 - sADMDialog 180
- Move
 - sADMDialog 182
 - sADMLItem 417

N

- NumberOfAllSelectedEntriesInHierarchy
 - sADMHierarchyList 321
- NumberOfEntries
 - sADMHierarchyList 319
 - sADMLList 473
- NumberOfExpandedEntries
 - sADMHierarchyList 321

- NumberOfLeafEntries
 - sADMHierarchyList 319
- NumberOfSelectedEntries
 - sADMHierarchyList 320
 - sADMLList 474
- NumberOfUnNestedSelectedEntriesInHierarchy
 - sADMHierarchyList 322

P

- PickEntry
 - sADMHierarchyList 322
 - sADMLList 474
- PickLeafEntry
 - sADMHierarchyList 323
- PluginAboutBox
 - sADMBasic 123
- PopupModal
 - sADMDialog 182

Q

- QuestionAlert
 - sADMBasic 124

R

- RegisterItemType
 - sADMDialog 183
- ReleaseADMWindowPort
 - sADMDrawer 253
- ReleaseMouseCapture
 - sADMTracker 541
- RemoveEntry
 - sADMHierarchyList 323
 - sADMLList 475
- RemoveItem
 - sADMLItem 418

S

- ScreenToLocalPoint
 - sADMDialog 184
 - sADMEEntry 281

- sADMHierarchyList 324
- sADMItem 419
- sADMListEntry 513
- ScreenToLocalRect
 - sADMDialog 184
 - sADMDialog 282
 - sADMItem 419
 - sADMListEntry 513
- Select
 - sADMDialog 282
 - sADMListEntry 514
- SelectAll
 - sADMItem 420
- SelectByText
 - sADMList 475
- SendNotify
 - sADMDialog 185
 - sADMDialog 283
 - sADMItem 420
 - sADMListEntry 514
- SetADMColor
 - sADMDrawer 253
- SetAlertButtonText
 - sADMBasic 125
- SetAllowMath
 - sADMItem 421
- SetAllowUnits
 - sADMItem 421
- SetAppFPS
 - sADMBasic 136
- SetAppUnits
 - sADMBasic 136
- SetBackColor
 - sADMItem 422
- SetBackgroundColor
 - sADMHierarchyList 324
 - sADMList 475
 - sADMListEntry 515
- SetBooleanValue
 - sADMItem 422
- SetBoundsRect
 - sADMDialog 185
 - sADMItem 423
- SetCancelItemID
 - sADMDialog 186
- SetCheckGlyph
 - sADMDialog 284
- SetClipPolygon
 - sADMDrawer 254
- SetClipRect
 - sADMDrawer 254
- SetCursorID
 - sADMDialog 187
 - sADMItem 423
- SetDefaultIncrements
 - sADMBasic 137
- SetDefaultItemID
 - sADMDialog 188
- SetDestroyProc
 - sADMDialog 188
 - sADMHierarchyList 325
 - sADMItem 424
 - sADMList 476
- SetDestroyProcRecursive
 - sADMHierarchyList 326
- SetDialogGroupInfo
 - sADMDialogGroup 212
- SetDialogName
 - sADMDialog 189
- SetDialogStyle
 - sADMDialog 189
- SetDisabledPicture
 - sADMDialog 284
 - sADMItem 425
 - sADMListEntry 515
- SetDisabledPictureID
 - sADMDialog 285
 - sADMItem 425
 - sADMListEntry 516
- SetDivided
 - sADMHierarchyList 326
- SetDividedRecursive
 - sADMHierarchyList 327
- SetDividingLineColor
 - sADMListEntry 517
- SetDrawMode
 - sADMDrawer 255
- SetDrawProc

- sADMHierarchyList 327
- sADMItem 426
- sADMList 477
- SetDrawProcRecursive
 - sADMHierarchyList 328
- SetEntryHeight
 - sADMHierarchyList 329
 - sADMList 477
- SetEntryHeightRecursive
 - sADMHierarchyList 330
- SetEntryItem
 - sADMListEntry 517
- SetEntryTextRect
 - sADMHierarchyList 330
 - sADMList 478
 - sADMListEntry 518
- SetEntryTextRectRecursive
 - sADMHierarchyList 331
- SetEntryWidth
 - sADMHierarchyList 331
 - sADMList 478
- SetEntryWidthRecursive
 - sADMHierarchyList 332
- SetFixedValue
 - sADMItem 427
- SetFlags
 - sADMHierarchyList 333
- SetFlagsRecursive
 - sADMHierarchyList 333
- SetFloatToTextProc
 - sADMItem 428
- SetFloatValue
 - sADMItem 428
- SetFont
 - sADMDialog 191
 - sADMDrawer 255
 - sADMItem 429
 - sADMListEntry 518
- SetForcedOnScreen
 - sADMDialog 191
- SetForeColor
 - sADMItem 429
- SetHasRollOverProperty
 - sADMItem 430
- SetHelpID
 - sADMDialog 204
 - sADMListEntry 291
 - sADMItem 459
 - sADMListEntry 524
- SetHorizontalIncrement
 - sADMDialog 192
- SetID
 - sADMListEntry 285
 - sADMListEntry 518
- SetIndentationWidth
 - sADMHierarchyList 334
- SetIndentationWidthRecursive
 - sADMHierarchyList 335
- SetInitProc
 - sADMHierarchyList 335
 - sADMList 479
- SetInitProcRecursive
 - sADMHierarchyList 336
- SetInRollOverState
 - sADMItem 430
- SetIntValue
 - sADMItem 431
- SetItemStyle
 - sADMItem 431
- SetItemType
 - sADMItem 432
- SetJustify
 - sADMItem 433
- SetLargeIncrement
 - sADMItem 434
- SetLocalLeftMargin
 - sADMHierarchyList 337
- SetLocalRect
 - sADMDialog 192
 - sADMItem 434
- SetMask
 - sADMDialog 193
 - sADMHierarchyList 337
 - sADMItem 435
 - sADMList 480
- SetMaskRecursive
 - sADMHierarchyList 338
- SetMaxFixedValue

- sADMItem 435
- SetMaxFloatValue
 - sADMItem 436
- SetMaxHeight
 - sADMDialog 193
- SetMaxIntValue
 - sADMItem 436
- SetMaxTextLength
 - sADMItem 437
- SetMaxWidth
 - sADMDialog 194
- SetMenuID
 - sADMHierarchyList 339
 - sADMList 480
- SetMinFixedValue
 - sADMItem 437
- SetMinFloatValue
 - sADMItem 438
- SetMinHeight
 - sADMDialog 195
- SetMinIntValue
 - sADMItem 438
- SetMinWidth
 - sADMDialog 195
- SetNonLeafEntryTextRect
 - sADMHierarchyList 339
- SetNonLeafEntryTextRectRecursive
 - sADMHierarchyList 340
- SetNotifierData
 - sADMDialog 196
 - sADMHierarchyList 340
 - sADMItem 439
 - sADMList 481
- SetNotifyProc
 - sADMDialog 197
 - sADMHierarchyList 341
 - sADMItem 439
 - sADMList 481
- SetNotifyProcRecursive
 - sADMHierarchyList 342
- SetNumbersArePoints
 - sADMBasic 138
- SetOrigin
 - sADMDrawer 256
- SetPicture
 - sADMEEntry 286
 - sADMItem 440
 - sADMListEntry 519
- SetPictureID
 - sADMEEntry 286
 - sADMItem 441
 - sADMListEntry 519
- SetPixel
 - sADMImage 365
- SetPlatformCursor
 - sADMBasic 117
- SetPluginRef
 - sADMItem 442
- SetPopupDialog
 - sADMItem 442
- SetPrecision
 - sADMItem 443
- SetRGBColor
 - sADMDrawer 256
- SetRolloverPicture
 - sADMItem 444
- SetRolloverPictureID
 - sADMItem 444
- SetSelectedPicture
 - sADMEEntry 287
 - sADMItem 445
 - sADMListEntry 520
- SetSelectedPictureID
 - sADMEEntry 288
 - sADMItem 445
 - sADMListEntry 521
- SetSelectionRange
 - sADMItem 446
- SetSmallIncrement
 - sADMItem 447
- SetTabGroup
 - sADMDialogGroup 213
- SetText
 - sADMDialog 197
 - sADMEEntry 288
 - sADMItem 448
 - sADMListEntry 521
- SetTextColor

- sADMListEntry 522
- SetTextToFloatProc
 - sADMItem 449
- SetTipString
 - sADMItem 459
- SetTrackProc
 - sADMDialog 198
 - sADMHierarchyList 343
 - sADMItem 450
 - sADMList 482
- SetTrackProcRecursive
 - sADMHierarchyList 344
- SetUnits
 - sADMItem 451
- SetUpdateEnabled
 - sADMDialog 199
- SetUserData
 - sADMDialog 199
 - sADMEEntry 289
 - sADMHierarchyList 344
 - sADMItem 452
 - sADMList 483
 - sADMListEntry 522
- SetVerticalIncrement
 - sADMDialog 200
- SetWantsFocus
 - sADMItem 452
- SetWindowRef
 - sADMDialog 200
- Show
 - sADMDialog 201
 - sADMItem 453
- ShowAllFloatingDialogs
 - sADMDialogGroup 214
- ShowToolTip
 - sADMBasic 125
 - sADMItem 460
- ShowUnits
 - sADMItem 453
- Size
 - sADMDialog 202
 - sADMItem 454
- SkipNextClipboardOperation
 - sADMNotifier 535

- StandardGetDirectoryDialog
 - sADMBasic 128
- StandardGetFileDialog
 - sADMBasic 126
- StandardPutFileDialog
 - sADMBasic 129
- StartMultipleItemInvalidate
 - sADMHierarchyList 345
- StopMultipleItemInvalidate
 - sADMHierarchyList 346
- StringToValue
 - sADMBasic 138
- SubtractClipPolygon
 - sADMDrawer 257
- SubtractClipRect
 - sADMDrawer 258
- SwapEntries
 - sADMHierarchyList 346

T

- TestAction
 - sADMTracker 541
- TestModifier
 - sADMTracker 542
- ToggleAllButNoCloseFloatingDialogs
 - sADMDialogGroup 214
- ToggleAllFloatingDialogs
 - sADMDialogGroup 214

U

- UnionClipPolygon
 - sADMDrawer 258
- UnionClipRect
 - sADMDrawer 259
- UnlinkEntry
 - sADMHierarchyList 347
- UnregisterItemType
 - sADMDialog 202
- Update
 - sADMDialog 203
 - sADMEEntry 290
 - sADMItem 454

sADMListEntry 523

V

ValueToString
sADMBasic 139

W

WasPercentageChange
sADMItem 455

Y

YesNoAlert
sADMBasic 130