Contents

Preface .................................................. 23
  Introduction ........................................ 23
  Conventions ........................................ 23
  Accessing Suites ................................... 24
  Supporting Documents ............................... 24

Chapter 1    ADM Overview ............................... 25
  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 ................................................ 29
  Events .............................................. 32
  Properties ......................................... 32
  Resources .......................................... 34
  The Suites ......................................... 35
  ADM Object Specifics ............................... 36
    ADM Dialog Objects .............................. 36
    Dialog Item Objects ............................ 39
    Composite Items ................................ 51
    ADM Item Groups ................................ 51
    ADM Item Numeric Properties .................. 52
    ADM Lists and Entries ........................... 52
    ADM Hierarchy Lists and List Entries ........ 54
    Macintosh and Windows ADM Item Resource Lists. 54
  Event Callbacks .................................... 62
    Using Event Callbacks ........................... 62
    Using Init Functions ............................ 63
    Using Drawer Functions .......................... 64
    Using Notifier Functions ....................... 65
    Using Tracker Functions ....................... 66
    Using Destroy Functions ....................... 67
    Using Resizable Windows ....................... 67
## Contents

Custom Item Types.......................................................................................... 68
Using Custom Items.................................................................................... 69
Using Timer Procedures............................................................................... 69
Using the C++ Interfaces............................................................................ 70
Getting Started With ADM Plug-In Development.................................. 71
  The General Development Process ......................................................... 71

### Chapter 2 Using ADM with Adobe Acrobat ............................................. 77

Introduction................................................................................................. 77
Using ADM with Acrobat........................................................................... 77
  Initializing ADM ...................................................................................... 78
  Acquiring and Using ADM Suites .......................................................... 78
  Using Modal Dialogs ............................................................................... 80
  Handling Resources ............................................................................... 81
Macintosh Issues........................................................................................ 82
  Carbon .................................................................................................. 82
  ‘carb’ Resources .................................................................................. 82
  Development Environment and Documentation .................................... 82

### Chapter 3 Using ADM with Adobe Photoshop ...................................... 83

Introduction................................................................................................. 83
Frame Select Photoshop Plug-in............................................................... 83
Platform-Specific Resources...................................................................... 83
Acquiring the Suites.................................................................................. 89
Building, Presenting, and Using the Dialog............................................. 90
  Operation ............................................................................................... 91

### Chapter 4 Using ADM with Adobe Illustrator ....................................... 101

A Modeless Dialog Example Using Illustrator ........................................ 101

### Chapter 5 Using ADM with Adobe After Effects ................................... 115

Introduction................................................................................................. 115
Implementation notes.................................................................................. 115
  Easy_Cheese Plug-in ............................................................................. 115

### Chapter 6 The ADM Basic Suite............................................................. 117
About the ADM Basic Suite .......................................................... 117
Accessing the Suite ................................................................. 117
ADM Basic Suite Functions ......................................................... 118
Resource Functions .................................................................. 118
  sADMBasic->GetIndexString() .................................................. 118
  sADMBasic->SetPlatformCursor() .............................................. 119
User Interface Functions .......................................................... 119
  sADMBasic->AboutBox() .......................................................... 120
  sADMBasic->AreToolTipsEnabled() ............................................. 120
  sADMBasic->AreToolTipsSticky() .............................................. 121
  sADMBasic->Beep() ................................................................. 121
  sADMBasic->ChooseColor() ...................................................... 122
  sADMBasic->EnableToolTips() ................................................... 122
  sADMBasic->ErrorAlert() ......................................................... 123
  sADMBasic->GetToolTipDelays() ............................................... 123
  sADMBasic->HideToolTip() ........................................................ 124
  sADMBasic->LightweightErrorAlert() ....................................... 124
  sADMBasic->MessageAlert() ..................................................... 125
  sADMBasic->PluginAboutBox() .................................................. 126
  sADMBasic->QuestionAlert() ..................................................... 126
  sADMBasic->SetAlertButtonText() ............................................. 127
  sADMBasic->ShowToolTip() ...................................................... 128
  sADMBasic->StandardGetFileDialog() ....................................... 129
  sADMBasic->StandardGetDirectoryDialog() ............................... 131
  sADMBasic->StandardPutFileDialog() ....................................... 132
  sADMBasic->YesNoAlert() ....................................................... 132
Utility Functions ..................................................................... 133
  sADMBasic->ADMColorToRGBColor() ........................................ 133
  sADMBasic->GetAppFPS() ........................................................ 134
  sADMBasic->GetAppUnits() ...................................................... 134
  sADMBasic->GetDefaultIncrements() ......................................... 135
  sADMBasic->GetLastADMError() .............................................. 136
  sADMBasic->GetNumbersArePoints() ........................................ 136
  sADMBasic->GetPaletteLayoutBounds() ..................................... 137
  sADMBasic->GetScreenDimensions() ......................................... 137
  sADMBasic->GetWorkspaceBounds() ......................................... 138
  sADMBasic->LookUpZString() .................................................. 138
  sADMBasic->SetAppFPS() ........................................................ 139
  sADMBasic->SetAppUnits() ...................................................... 139
  sADMBasic->SetDefaultIncrements() ......................................... 140
  sADMBasic->SetNumbersArePoints() ......................................... 141
  sADMBasic->StringValue() ...................................................... 141
  sADMBasic->ValueToString() .................................................... 142
Contextual Menu Functions ....................................................... 143
  sADMBasic->CreateMenu() ...................................................... 143
Chapter 7  The ADM Dialog Suite  

About the ADM Dialog Suite ................................. 145
Accessing the Suite ......................................... 145
Dialog Basics: Styles ....................................... 145
Dialog Basics: Standard Dialog Item IDs ................. 146
Dialog Basics: Callbacks .................................. 147
ADM Dialog Suite Functions ............................... 147
  sADMDialog->AbortTimer() .......................... 147
  sADMDialog->Activate() ............................... 148
  sADMDialog->AdjustItemTabOrder() .................. 148
  sADMDialog->Create() .................................. 149
  sADMDialog->CreateGroupInDialog() ................. 152
  sADMDialog->CreateItem() ............................. 153
  sADMDialog->CreateTimer() ............................ 154
  sADMDialog->DefaultDraw() ............................ 155
  sADMDialog->DefaultNotify() ......................... 156
  sADMDialog->DefaultTrack() ........................... 156
  sADMDialog->Destroy() .................................. 157
  sADMDialog->DestroyItem() ............................. 158
  sADMDialog->DisplayAsModal() ......................... 159
  sADMDialog->DisplayAsPopupModal() .................. 159
  sADMDialog->Enable() .................................... 160
  sADMDialog->EndModal() ................................ 160
  sADMDialog->GetBoundsRect() ......................... 161
  sADMDialog->GetCancelItemID() ....................... 162
  sADMDialog->GetCursorID() ............................. 162
  sADMDialog->GetDefaultItemID() ....................... 163
  sADMDialog->GetDestroyProc() ....................... 163
  sADMDialog->GetDialogName() .......................... 164
  sADMDialog->GetDialogStyle() ......................... 164
  sADMDialog->GetDrawProc() ............................ 165
  sADMDialog->GetFont() ................................... 165
  sADMDialog->GetHorizontalIncrement() ............... 166
  sADMDialog->GetID() .................................... 167
  sADMDialog->GetItem() ................................... 167
  sADMDialog->GetLocalRect() ............................ 168
  sADMDialog->GetMask() ................................... 168
  sADMDialog->GetMaxHeight() ............................ 169
  sADMDialog->GetMaxWidth() ............................. 169
  sADMDialog->GetMinHeight() ............................ 170
  sADMDialog->GetMinWidth() ............................. 170

sADMDialog->GetMinWidth() ................................. 170
sADMDialog->GetMinHeight() ............................... 170
sADMDialog->GetMaxWidth() ................................. 169
sADMDialog->GetMaxHeight() ............................... 169
sADMDialog->GetLocalRect() ............................... 168
sADMDialog->GetMask()  ................................... 168
sADMDialog->GetDefaultItemID() .......................... 163
sADMDialog->GetDestroyProc() ........................... 163
sADMDialog->GetDialogName() ............................. 164
sADMDialog->GetDialogStyle() ............................ 164
sADMDialog->GetDrawProc() ............................... 165
sADMDialog->GetFont()  .................................... 165
sADMDialog->GetHorizontalIncrement() .................. 166
sADMDialog->GetID()  ....................................... 167
sADMDialog->GetItem()  ..................................... 167
sADMDialog->GetLocalRect() .............................. 168
sADMDialog->GetMask()  ..................................... 168
sADMDialog->GetMaxHeight() .............................. 169
sADMDialog->GetMaxWidth() ............................... 169
sADMDialog->GetMinHeight() .............................. 170
sADMDialog->GetMinWidth() ............................... 170
sADMDialog->GetNextItem() ........................................... 171
sADMDialog->GetNotifierData() .................................. 172
sADMDialog->GetNotifyProc() ...................................... 172
sADMDialog->GetPluginRef() ......................................... 173
sADMDialog->GetPreviousItem() .................................... 173
sADMDialog->GetText() ................................................ 174
sADMDialog->GetTextLength() ....................................... 174
sADMDialog->GetTrackProc() ......................................... 175
sADMDialog->GetUserData() .......................................... 175
sADMDialog->GetVerticalIncrement() ............................... 176
sADMDialog->GetWindowRef() ....................................... 176
sADMDialog->Invalidate() ............................................. 177
sADMDialog->InvalidateRect() ....................................... 178
sADMDialog->IsVisible() .............................................. 178
sADMDialog->IsDialogContextHidden() ......................... 179
sADMDialog->IsEnabled() ............................................. 180
sADMDialog->IsForcedOnScreen() .................................. 180
sADMDialog->IsUpdateEnabled() .................................... 181
sADMDialog->Move() ................................................... 183
sADMDialog->NextItem() ............................................. 183
sADMDialog->PopupModal() ......................................... 185
sADMDialog->PreviousItem() ........................................ 185
sADMDialog->RegisterItemType() ................................. 186
sADMDialog->ScreenToLocalPoint() ............................... 187
sADMDialog->ScreenToLocalRect() ................................. 187
sADMDialog->SendNotify() .......................................... 188
sADMDialog->SetBoundsRect() ..................................... 188
sADMDialog->SetCancellableID() ................................. 189
sADMDialog->SetCursorID() ......................................... 190
sADMDialog->SetDefaultItemID() ................................ 191
sADMDialog->SetDestroyProc() .................................... 191
sADMDialog->SetDialogContextHidden() ....................... 191
sADMDialog->SetDialogName() .................................... 192
sADMDialog->SetDialogStyle() .................................... 192
sADMDialog->SetDrawProc() ....................................... 193
sADMDialog->SetFont() .............................................. 194
sADMDialog->SetForcedOnScreen() ............................... 194
sADMDialog->SetHorizontalIncrement() ......................... 195
sADMDialog->SetLocalRect() ....................................... 195
sADMDialog->SetMask() .............................................. 196
sADMDialog->SetMaxHeight() ....................................... 197
sADMDialog->SetMaxWidth() ....................................... 197
sADMDialog->SetMinHeight() ....................................... 198
sADMDialog->SetMinWidth() ....................................... 199
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sADMDialog-&gt;SetNotifyProc()</td>
<td>200</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetText()</td>
<td>201</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetTrackProc()</td>
<td>201</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetUpdateEnabled()</td>
<td>202</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetUserData()</td>
<td>203</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetVerticalIncrement()</td>
<td>203</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetWindowRef()</td>
<td>204</td>
</tr>
<tr>
<td>sADMDialog-&gt;Show()</td>
<td>205</td>
</tr>
<tr>
<td>sADMDialog-&gt;Size()</td>
<td>205</td>
</tr>
<tr>
<td>sADMDialog-&gt;UnregisterItemType()</td>
<td>206</td>
</tr>
<tr>
<td>sADMDialog-&gt;GetHelpID()</td>
<td>207</td>
</tr>
<tr>
<td>sADMDialog-&gt;Help()</td>
<td>207</td>
</tr>
<tr>
<td>sADMDialog-&gt;Update()</td>
<td>206</td>
</tr>
<tr>
<td>sADMDialog-&gt;UnregisterItemType()</td>
<td>206</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetWindowRef()</td>
<td>204</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetVerticalIncrement()</td>
<td>203</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetUserData()</td>
<td>203</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetUpdateEnabled()</td>
<td>202</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetTrackProc()</td>
<td>201</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetText()</td>
<td>201</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetNotifyProc()</td>
<td>200</td>
</tr>
<tr>
<td>sADMDialog-&gt;SetWindowRef()</td>
<td>204</td>
</tr>
<tr>
<td>sADMDialog-&gt;Show()</td>
<td>205</td>
</tr>
<tr>
<td>sADMDialog-&gt;Size()</td>
<td>205</td>
</tr>
<tr>
<td>sADMDialog-&gt;UnregisterItemType()</td>
<td>206</td>
</tr>
</tbody>
</table>

**Chapter 8**  The ADM Dialog Group Suite ........................ 209

About the ADM Dialog Group Suite .................................. 209
Accessing the Suite .................................................. 209
The ADM Dialog Group Suite’s Position Code and Group Name .. 209
ADM DialogGroup Suite Functions ................................... 210
sADMDialogGroup->CountDialogs() ......................... 210
sADMDialogGroup->GetDialogGroupInfo() ................. 210
sADMDialogGroup->GetDialogName() ...................... 211
sADMDialogGroup->GetNamedDialog() .................... 211
sADMDialogGroup->GetNthDialog() ....................... 212
sADMDialogGroup->IsCollapsed() ....................... 212
sADMDialogGroup->IsDockVisible() ..................... 213
sADMDialogGroup->IsFrontTab() ......................... 214
sADMDialogGroup->IsStandAlonePalette() .............. 214
sADMDialogGroup->SetDialogGroupInfo() .................. 215
sADMDialogGroup->SetTabGroup() ....................... 215
sADMDialogGroup->ShowAllFloatingDialogs() .......... 216
sADMDialogGroup->ToggleAllButNoCloseFloatingDialogs() | 216
sADMDialogGroup->ToggleAllFloatingDialogs() ........ 217

**Chapter 9**  The ADM Drawer Suite ............................... 219

About the ADM Drawer Suite ....................................... 219
Accessing the Suite ................................................. 219
ADM Drawer Functions ............................................. 220
Using ADM Drawer Functions .................................... 220
Fonts and Colors .......................................................... 221
The Drawer Coordinate Space ........................................ 221
Drawing Modes ............................................................ 222
ADM Drawer Suite Functions .......................................... 223
  sADMDrawer->GetOrigin() ............................................. 223
  sADMDrawer->GetFontInfo() .......................................... 223
  sADMDrawer->GetFont() ............................................... 224
  sADMDrawer->GetDrawMode() ........................................ 224
  sADMDrawer->GetClipRect() ......................................... 224
  sADMDrawer->GetBoundsRect() ...................................... 224
  sADMDrawer->GetAGMPort() .......................................... 225
  sADMDrawer->GetADMWindowPort() ................................. 225
  sADMDrawer->GetADMColor() ...................................... 226
  sADMDrawer->FillRect() .............................................. 226
  sADMDrawer->FillPolygon() ......................................... 227
  sADMDrawer->FillOval() .............................................. 227
  sADMDrawer->DrawUpArrow() ........................................ 228
  sADMDrawer->DrawTextRight() ........................................ 229
  sADMDrawer->DrawTextLeft() ......................................... 229
  sADMDrawer->DrawTextInaBox() .................................... 230
  sADMDrawer->DrawTextCentered() .................................. 231
  sADMDrawer->DrawText() ............................................. 232
  sADMDrawer->DrawSunkenRect() .................................... 233
  sADMDrawer->DrawResPicture() ...................................... 234
  sADMDrawer->DrawResPictureCentered() ............................ 235
  sADMDrawer->DrawRect() ............................................ 236
  sADMDrawer->DrawResPicture() ...................................... 237
  sADMDrawer->DrawResPictureCentered() ............................ 238
  sADMDrawer->DrawRightArrow() ...................................... 239
  sADMDrawer->DrawSunkenRect() .................................... 239
  sADMDrawer->DrawText() ............................................. 240
  sADMDrawer->DrawTextCentered() .................................. 241
  sADMDrawer->DrawTextInaBox() .................................... 242
  sADMDrawer->DrawTextRight() ........................................ 242
  sADMDrawer->DrawUpArrow() ........................................ 243
  sADMDrawer->FillOval() .............................................. 244
  sADMDrawer->FillPolygon() ......................................... 245
  sADMDrawer->FillRect() .............................................. 245
  sADMDrawer->GetADMColor() ...................................... 246
  sADMDrawer->GetAGMPort() .......................................... 247
  sADMDrawer->GetBoundsRect() ...................................... 247
  sADMDrawer->GetClipRect() ......................................... 248
  sADMDrawer->GetDrawModel() ....................................... 248
  sADMDrawer->GetFont() .............................................. 249
  sADMDrawer->GetFontInfo() .......................................... 249
  sADMDrawer->GetOrigin() ............................................ 250
Chapter 10  The ADM Entry Suite  265

About the ADM Entry Suite  265
Accessing the Suite  265
Initializing An Entry  265

ADM Entry Suite Functions  266
sADMEntry->AbortTimer()  266
sADMEntry->Activate()  266
sADMEntry->Check()  267
sADMEntry->Create()  267
sADMEntry->CreateTimer()  268
sADMEntry->DefaultDraw()  269
sADMEntry->DefaultNotify()  270
sADMEntry->DefaultTrack()  270
sADMEntry->Destroy()  271
sADMEntry->Enable()  272
sADMEntry->GetBoundsRect()  273
sADMEntry->GetCheckGlyph()  273
sADMEntry->GetDisabledPicture()  274
sADMEntry->GetDisabledPictureID()  274
sADMEntry->GetID()  275
sADMEntry->GetIndex()  275
sADMEntry->GetList()  276
Chapter 11 The ADM Hierarchy List Suite

About the ADM Hierarchy List Suite
Accessing the Suite
ADM Hierarchy Lists and List Entries
ADM Hierarchy List Recipes
Custom ADM Hierarchy Lists .............................................................. 301

ADM Hierarchy List Suite Functions .............................................. 302

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sADMHierarchyList-&gt;DeselectAll()</td>
<td>302</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;FindEntry()</td>
<td>302</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetActiveEntry()</td>
<td>303</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetActiveLeafEntry()</td>
<td>303</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetDestroyProc()</td>
<td>304</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetDivided()</td>
<td>304</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetDrawProc()</td>
<td>305</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetEntry()</td>
<td>305</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetEntryHeight()</td>
<td>306</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetEntryTextRect()</td>
<td>306</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetEntryWidth()</td>
<td>307</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetExpandedIndex()</td>
<td>308</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetFlags()</td>
<td>309</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetGlobalLeftMargin()</td>
<td>309</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetIndentationWidth()</td>
<td>309</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetInitProc()</td>
<td>310</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetItem()</td>
<td>310</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetLeafIndex()</td>
<td>311</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetLocalLeftMargin()</td>
<td>311</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetLocalRect()</td>
<td>312</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetMask()</td>
<td>312</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetMenuID()</td>
<td>312</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetNonLeafEntryHeight()</td>
<td>313</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetNonLeafEntryTextRect()</td>
<td>313</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetNotifierData()</td>
<td>314</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetNotifyProc()</td>
<td>314</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetParentEntry()</td>
<td>315</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetTrackProc()</td>
<td>315</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GetUserData()</td>
<td>316</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GlobalToLocalPoint()</td>
<td>317</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;GlobalToLocalRect()</td>
<td>317</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;IndexOfAllSelectedEntriesInHierarchy()</td>
<td>318</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;IndexOfEntry()</td>
<td>319</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;IndexOfExpandedEntry()</td>
<td>319</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;IndexOfLeafEntry()</td>
<td>320</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;IndexOfSelectedEntry()</td>
<td>320</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;IndexOfUnNestedSelectedEntriesInHierarchy()</td>
<td>321</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;InsertEntry()</td>
<td>322</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;InsertGivenEntry()</td>
<td>322</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;Invalidate()</td>
<td>323</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;LocalToLocalPoint()</td>
<td>323</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;LocalToLocalRect()</td>
<td>324</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;LocalToScreenPoint()</td>
<td>325</td>
</tr>
<tr>
<td>sADMHierarchyList-&gt;NumberOfEntries()</td>
<td>325</td>
</tr>
</tbody>
</table>
sADMHierarchyList->NumberOfLeafEntries() ............................................. 326
sADMHierarchyList->NumberOfSelectedEntries() ................................... 326
sADMHierarchyList->NumberOfAllSelectedEntriesInHierarchy() .......... 327
sADMHierarchyList->NumberOfExpandedEntriesInHierarchy() .......... 328
sADMHierarchyList->
NumberOfUnNestedSelectedEntriesInHierarchy() .................................. 328
sADMHierarchyList->PickEntry() ...................................................... 329
sADMHierarchyList->PickLeafEntry() ............................................... 329
sADMHierarchyList->RemoveEntry() .................................................. 330
sADMHierarchyList->ScreenToLocalPoint() ........................................... 330
sADMHierarchyList->SetBackgroundColor() ......................................... 331
sADMHierarchyList->SetDestroyProc() ............................................... 331
sADMHierarchyList->SetDestroyProcRecursive() ................................. 332
sADMHierarchyList->SetDivided() .................................................... 333
sADMHierarchyList->SetDividedRecursive() ....................................... 334
sADMHierarchyList->SetDrawProc() .................................................. 334
sADMHierarchyList->SetDrawProcRecursive() ..................................... 335
sADMHierarchyList->SetEntryHeight() ............................................. 336
sADMHierarchyList->SetEntryHeightRecursive() .................................. 337
sADMHierarchyList->SetEntryTextRect() ............................................ 337
sADMHierarchyList->SetEntryTextRectRecursive() ............................... 338
sADMHierarchyList->SetEntryWidth() .............................................. 338
sADMHierarchyList->SetEntryWidthRecursive() .................................. 339
sADMHierarchyList->SetFlags() ...................................................... 340
sADMHierarchyList->SetFlagsRecursive() ......................................... 341
sADMHierarchyList->SetIndentationWidth() ....................................... 341
sADMHierarchyList->SetIndentationWidthRecursive() ......................... 342
sADMHierarchyList->SetInitProc() ................................................... 343
sADMHierarchyList->SetInitProcRecursive() ....................................... 343
sADMHierarchyList->SetLocalLeftMargin() ....................................... 344
sADMHierarchyList->SetMask() ...................................................... 345
sADMHierarchyList->SetMaskRecursive() .......................................... 345
sADMHierarchyList->SetMenuID() .................................................... 346
sADMHierarchyList->SetNonLeafEntryTextRect() ................................ 347
sADMHierarchyList->SetNonLeafEntryTextRectRecursive() .................... 347
sADMHierarchyList->SetNotifierData() ............................................ 348
sADMHierarchyList->SetNotifyProc() .............................................. 348
sADMHierarchyList->SetNotifyProcRecursive() .................................. 349
sADMHierarchyList->SetTrackProc() ................................................ 350
sADMHierarchyList->SetTrackProcRecursive() .................................... 351
sADMHierarchyList->SetUserData() ................................................. 352
sADMHierarchyList->StartMultipleItemInvalidate() ............................ 353
sADMHierarchyList->StopMultipleItemInvalidate() ............................. 353
sADMHierarchyList->SwapEntries() .................................................. 354
sADMHierarchyList->UnlinkEntry() ................................................... 354
Chapter 12   The ADM Icon Suite .............................................. 357
About the ADM Icon Suite ................................................ 357
Accessing the Suite .......................................................... 357
ADM Icons ........................................................................ 357
ADM Icon Suite Functions ..................................................... 358
  sADMIcon->Create() ......................................................... 358
  sADMIcon->CreateFromImage() ......................................... 359
  sADMIcon->Destroy() ......................................................... 359
  sADMIcon->GetData() ......................................................... 360
  sADMIcon->GetFromResource() ......................................... 360
  sADMIcon->GetHeight() ...................................................... 361
  sADMIcon->GetType() ......................................................... 362
  sADMIcon->GetWidth() ...................................................... 362
  sADMIcon->IsFromResource() ........................................... 363

Chapter 13   The ADM Image Suite .......................................... 365
About the ADM Image Suite .................................................. 365
Accessing the Suite ............................................................ 365
ADM Image Suite Functions .................................................. 365
  sADMImage->BeginADMDrawer() ......................................... 365
  sADMImage->BeginAGMImageAccess() ................................ 366
  sADMImage->BeginBaseAddressAccess() .............................. 367
  sADMImage->Create() ....................................................... 367
  sADMImage->CreateBitmap() ............................................. 368
  sADMImage->CreateOffscreen() ......................................... 368
  sADMImage->Destroy() ....................................................... 369
  sADMImage->EndAGMImageAccess() .................................... 369
  sADMImage->EndBaseAddressAccess() ................................ 370
  sADMImage->EndADMDrawer() ............................................ 370
  sADMImage->GetBitsPerPixel() .......................................... 371
  sADMImage->GetHeight() .................................................. 371
  sADMImage->GetWidth() .................................................. 372
  sADMImage->GetPixel() .................................................... 372
  sADMImage->SetPixel() ..................................................... 373

Chapter 14   The ADM Item Suite ............................................ 375
About the ADM Item Suite .................................................... 375
Accessing the Suite ........................................................... 375
Initializing An Item ............................................................ 375
FloatToText and TextToFloat Functions ................................. 379
ADM Help Support ................................................................. 380
ADM Item Suite Functions ...................................................... 381
  sADMItem->AbortTimer() ...................................................... 381
  sADMItem->Activate() ........................................................ 382
  sADMItem->AddItem() ........................................................ 382
  sADMItem->Create() .......................................................... 383
  sADMItem->CreateTimer() ..................................................... 384
  sADMItem->DefaultDraw() .................................................... 385
  sADMItem->DefaultFloatToText() ......................................... 386
  sADMItem->DefaultTextToFloat() ......................................... 387
  sADMItem->DefaultNotify() .................................................. 387
  sADMItem->DefaultTrack() ................................................... 388
  sADMItem->Destroy() .......................................................... 389
  sADMItem->Enable() ............................................................. 390
  sADMItem->GetAllowMath() .................................................. 390
  sADMItem->GetAllowUnits() .................................................. 391
  sADMItem->GetBackColor() ................................................... 391
  sADMItem->GetBestSize() ...................................................... 392
  sADMItem->GetBooleanValue() .............................................. 392
  sADMItem->GetBoundsRect() .................................................. 393
  sADMItem->GetChildItem() .................................................... 393
  sADMItem->GetCursorID() ..................................................... 395
  sADMItem->GetDestroyProc() ................................................. 395
  sADMItem->GetDialog() ........................................................ 396
  sADMItem->GetDisabledPicture() .......................................... 396
  sADMItem->GetDisabledPictureID() ....................................... 397
  sADMItem->GetDrawProc() .................................................... 397
  sADMItem->GetFixedValue() .................................................. 398
  sADMItem->GetFloatTextProc() ............................................. 398
  sADMItem->GetFloatValue() .................................................. 399
  sADMItem->GetFont() .......................................................... 399
  sADMItem->GetForeColor() .................................................... 400
  sADMItem->GetHasRolloverProperty() ..................................... 400
  sADMItem->GetHierarchyList() ............................................. 401
  sADMItem->GetID() .............................................................. 401
  sADMItem->GetIntValue() ...................................................... 401
  sADMItem->GetItemStyle() .................................................... 402
  sADMItem->GetItemCount() ................................................... 402
  sADMItem->GetItemType() ..................................................... 402
  sADMItem->GetJustify() ....................................................... 403
  sADMItem->GetLargeIncrement() .......................................... 403
  sADMItem->GetList() .......................................................... 404
  sADMItem->GetLocalRect() ................................................... 405
  sADMItem->GetMask() .......................................................... 405
  sADMItem->GetMaxFixedValue() ............................................. 406
  sADMItem->GetMaxFloatValue() ............................................. 406
  sADMItem->GetMaxIntValue() ................................................ 406

Adobe Dialog Manager Programmer's Guide and Reference
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sADMItem-&gt;GetMaxTextLength()</td>
<td>407</td>
</tr>
<tr>
<td>sADMItem-&gt;GetMinFixedValue()</td>
<td>407</td>
</tr>
<tr>
<td>sADMItem-&gt;GetMinFloatValue()</td>
<td>408</td>
</tr>
<tr>
<td>sADMItem-&gt;GetMinIntValue()</td>
<td>408</td>
</tr>
<tr>
<td>sADMItem-&gt;GetNotifierData()</td>
<td>409</td>
</tr>
<tr>
<td>sADMItem-&gt;GetNotifyProc()</td>
<td>409</td>
</tr>
<tr>
<td>sADMItem-&gt;GetPicture()</td>
<td>410</td>
</tr>
<tr>
<td>sADMItem-&gt;GetPictureID()</td>
<td>411</td>
</tr>
<tr>
<td>sADMItem-&gt;GetPluginRef()</td>
<td>411</td>
</tr>
<tr>
<td>sADMItem-&gt;GetPopupDialog()</td>
<td>412</td>
</tr>
<tr>
<td>sADMItem-&gt;GetPrecision()</td>
<td>412</td>
</tr>
<tr>
<td>sADMItem-&gt;GetRolloverPicture()</td>
<td>413</td>
</tr>
<tr>
<td>sADMItem-&gt;GetRolloverPictureID()</td>
<td>413</td>
</tr>
<tr>
<td>sADMItem-&gt;GetSelectedPicture()</td>
<td>414</td>
</tr>
<tr>
<td>sADMItem-&gt;GetSelectedPictureID()</td>
<td>415</td>
</tr>
<tr>
<td>sADMItem-&gt;GetSelectionRange()</td>
<td>415</td>
</tr>
<tr>
<td>sADMItem-&gt;GetShowUnits()</td>
<td>416</td>
</tr>
<tr>
<td>sADMItem-&gt;GetSmallIncrement()</td>
<td>416</td>
</tr>
<tr>
<td>sADMItem-&gt;GetText()</td>
<td>417</td>
</tr>
<tr>
<td>sADMItem-&gt;GetTextColor()</td>
<td>418</td>
</tr>
<tr>
<td>sADMItem-&gt;GetTextToFloatProc()</td>
<td>418</td>
</tr>
<tr>
<td>sADMItem-&gt;GetTrackProc()</td>
<td>418</td>
</tr>
<tr>
<td>sADMItem-&gt;GetUnits()</td>
<td>419</td>
</tr>
<tr>
<td>sADMItem-&gt;GetUserData()</td>
<td>419</td>
</tr>
<tr>
<td>sADMItem-&gt;GetWantsFocus()</td>
<td>420</td>
</tr>
<tr>
<td>sADMItem-&gt;GetWindowRef()</td>
<td>420</td>
</tr>
<tr>
<td>sADMItem-&gt;IgnoreForceRoman()</td>
<td>421</td>
</tr>
<tr>
<td>sADMItem-&gt;Invalidate()</td>
<td>422</td>
</tr>
<tr>
<td>sADMItem-&gt;InvalidateRect()</td>
<td>422</td>
</tr>
<tr>
<td>sADMItem-&gt;IsActive()</td>
<td>423</td>
</tr>
<tr>
<td>sADMItem-&gt;IsEnabled()</td>
<td>423</td>
</tr>
<tr>
<td>sADMItem-&gt;IsInRolloverState()</td>
<td>424</td>
</tr>
<tr>
<td>sADMItem-&gt;IsKnown()</td>
<td>424</td>
</tr>
<tr>
<td>sADMItem-&gt;IsVisible()</td>
<td>425</td>
</tr>
<tr>
<td>sADMItem-&gt;Known()</td>
<td>426</td>
</tr>
<tr>
<td>sADMItem-&gt;LocalToScreenPoint()</td>
<td>426</td>
</tr>
<tr>
<td>sADMItem-&gt;LocalToScreenRect()</td>
<td>427</td>
</tr>
<tr>
<td>sADMItem-&gt;Move()</td>
<td>427</td>
</tr>
<tr>
<td>sADMItem-&gt;RemoveItem()</td>
<td>428</td>
</tr>
<tr>
<td>sADMItem-&gt;ScreenToLocalPoint()</td>
<td>429</td>
</tr>
<tr>
<td>sADMItem-&gt;ScreenToLocalRect()</td>
<td>429</td>
</tr>
<tr>
<td>sADMItem-&gt;SelectAll()</td>
<td>430</td>
</tr>
<tr>
<td>sADMItem-&gt;SendNotify()</td>
<td>430</td>
</tr>
<tr>
<td>sADMItem-&gt;SetAllowMath()</td>
<td>431</td>
</tr>
<tr>
<td>sADMItem-&gt;SetAllowUnits()</td>
<td>431</td>
</tr>
<tr>
<td>sADMItem-&gt;SetBackColor()</td>
<td>432</td>
</tr>
<tr>
<td>sADMItem-&gt;SetBooleanValue()</td>
<td>432</td>
</tr>
</tbody>
</table>
Contents

sADMItem->SetBoundsRect() .............................................................. 433
sADMItem->SetCursorID() ............................................................... 434
sADMItem->SetDestroyProc() ............................................................ 435
sADMItem->SetDisabledPicture() ......................................................... 435
sADMItem->SetDisabledPictureID() ....................................................... 436
sADMItem->SetDrawProc() ................................................................. 437
sADMItem->SetFixedValue() ................................................................. 437
sADMItem->SetFloatToTextProc() ......................................................... 438
sADMItem->SetFloatValue() ................................................................. 439
sADMItem->SetFont() ........................................................................ 439
sADMItem->SetForeColor() ................................................................. 440
sADMItem->SetFont() ........................................................................ 440
sADMItem->SetHasRolloverProperty() ................................................... 440
sADMItem->SetInRolloverState() .......................................................... 441
sADMItem->SetIntValue() ................................................................. 442
sADMItem->SetItemType() ................................................................. 442
sADMItem->SetJustify() ................................................................. 443
sADMItem->SetLargeIncrement() ........................................................ 444
sADMItem->SetLocalRect() .............................................................. 445
sADMItem->SetMask() ...................................................................... 446
sADMItem->SetMaxFixedValue() ......................................................... 446
sADMItem->SetMaxFloatValue() ......................................................... 447
sADMItem->SetMaxIntValue() ............................................................. 447
sADMItem->SetMaxTextLength() ........................................................ 448
sADMItem->SetMinFixedValue() ........................................................ 448
sADMItem->SetMinFloatValue() ........................................................ 449
sADMItem->SetMinIntValue() ............................................................ 449
sADMItem->SetNotifierData() ........................................................... 450
sADMItem->SetNotifyProc() ............................................................. 451
sADMItem->SetPicture() ................................................................. 451
sADMItem->SetPictureID() ............................................................... 452
sADMItem->SetPluginRef() ............................................................... 453
sADMItem->SetPopupDialog() ........................................................... 453
sADMItem->SetPrecision() ............................................................... 454
sADMItem->SetRolloverPicture() ........................................................ 455
sADMItem->SetRolloverPictureID() ..................................................... 455
sADMItem->SetSelectedPicture() ........................................................ 456
sADMItem->SetSelectedPictureID() ..................................................... 457
sADMItem->SetSelectionRange() ........................................................ 458
sADMItem->SetSmallIncrement() ......................................................... 458
sADMItem->SetText() ...................................................................... 459
sADMItem->SetTextToFloatProc() ....................................................... 461
sADMItem->SetTrackProc() ............................................................... 461
sADMItem->SetUnits() ................................................................. 462
sADMItem->SetUserData() ............................................................... 462
sADMItem->SetWantsFocus() ............................................................. 463
sADMItem->Show() ..................................................................... 464
Chapter 15  The ADM List Suite  ................................................. 473

About the ADM List Suite ....................................................... 473
Accessing the Suite .............................................................. 473
ADM Lists and Entries ........................................................... 473
ADM List Recipes ................................................................. 474
Custom ADM Lists ............................................................... 474
ADM List Suite Functions ....................................................... 475

sADMList->FindEntry() .......................................................... 475
sADMList->GetActiveEntry() .................................................. 475
sADMList->GetDestroyProc() .................................................. 476
sADMList->GetDrawProc() ...................................................... 476
sADMList->GetEntry() ............................................................ 477
sADMList->GetEntryHeight() ................................................... 478
sADMList->GetEntryTextRect() ............................................... 478
sADMList->GetEntryWidth() .................................................... 479
sADMList->GetInitProc() ........................................................ 479
sADMList->GetItem() .............................................................. 480
sADMList->GetMask() ............................................................ 480
sADMList->GetMenuID() ........................................................ 480
sADMList->GetNotifierData() ................................................... 481
sADMList->GetNotifyProc() ..................................................... 481
sADMList->GetTrackProc() ...................................................... 482
sADMList->GetUserData() ....................................................... 483
sADMList->IndexEntry() ........................................................ 483
sADMList->IndexSelectedEntry() .............................................. 484
sADMList->InsertEntry() ......................................................... 485
sADMList->NumberOfEntries() ................................................. 485
sADMList->NumberOfSelectedEntries() ..................................... 486

sADMItem->ShowToolTip() ....................................................... 464
sADMItem->Size() ................................................................. 465
sADMItem->Update() ............................................................. 465
sADMItem->WasPercentageChange() ......................................... 466

ADM Help Support ............................................................... 467
sADMItem->EnableTip() .......................................................... 467
sADMItem->GetHelpID() ........................................................ 467
sADMItem->GetTipString() ...................................................... 468
sADMItem->GetTipStringLength() ............................................. 468
sADMItem->Help() ............................................................... 469
sADMItem->HideToolTip() ....................................................... 469
sADMItem->IsTipEnabled() ....................................................... 470
sADMItem->ShowToolTip() ...................................................... 470
sADMItem->SetTipString() ...................................................... 471
sADMItem->SetHelpID() ........................................................ 471
sADMItem->GetTipStringLength() ............................................. 468
sADMItem->GetTipString() ...................................................... 468
sADMItem->GetHelpID() ........................................................ 467
sADMItem->EnableTip() .......................................................... 467
sADMItem->WasPercentageChange() ......................................... 466
sADMItem->Update() ............................................................. 465
sADMItem->Size() ................................................................. 465
sADMItem->ShowUnits() ......................................................... 464
Chapter 16   The ADM List Entry Suite  ........................................ 497

About the ADM List Entry Suite ................................................... 497
ADM List Objects and Entries
    ADM Hierarchy List Objects and List Entries ........................... 498
ADM List Entry Suite Functions ................................................... 498
    sADMListEntry->AbortTimer() ............................................ 498
    sADMListEntry->Activate() .................................................. 499
    sADMListEntry->AreChildrenSelected() ................................ 499
    sADMListEntry->Check() ...................................................... 500
    sADMListEntry->Create() ..................................................... 501
    sADMListEntry->CreateChildList() ........................................ 501
    sADMListEntry->CreateTimer() ............................................... 501
    sADMListEntry->DefaultDraw() .............................................. 503
    sADMListEntry->DefaultNotify() .......................................... 504
    sADMListEntry->DefaultTrack() .......................................... 504
    sADMListEntry->DeleteChildList() ......................................... 505
    sADMListEntry->Destroy() .................................................... 506
    sADMListEntry->Enable() .................................................... 506
    sADMListEntry->EnableChildSelection() .................................. 507
    sADMListEntry->ExpandHierarchy() ........................................ 507
    sADMListEntry->GetBoundsRect() ........................................... 508
    sADMListEntry->GetChildList() ............................................. 508
    sADMListEntry->GetDisabledPicture() .................................... 508
    sADMListEntry->GetDisabledPictureID() .................................. 509
    sADMListEntry->GetEntryItem() ............................................. 510
    sADMListEntry->GetExpandArrowLocalRect() ............................... 510
    sADMListEntry->GetHierarchyDepth() ...................................... 511
    sADMListEntry->GetID() ...................................................... 511
sADMListEntry->GetIndex() ......................................................... 512
sADMListEntry->GetItem() ......................................................... 512
sADMListEntry->GetList() ......................................................... 512
sADMListEntry->GetLocalRect() ................................................... 513
sADMListEntry->GetPicture() ...................................................... 513
sADMListEntry->GetPictureID() .................................................... 514
sADMListEntry->GetSelectedPicture() ........................................... 514
sADMListEntry->GetSelectedPictureID() ........................................ 515
sADMListEntry->GetText() .......................................................... 516
sADMListEntry->GetTextLength() .................................................. 516
sADMListEntry->GetUserData() ..................................................... 517
sADMListEntry->GetVisualHierarchyDepth() ..................................... 517
sADMListEntry->HideEntryName() ................................................ 518
sADMListEntry->Invalidate() ....................................................... 519
sADMListEntry->IsActive() .......................................................... 520
sADMListEntry->IsChecked() ....................................................... 520
sADMListEntry->IsChildSelectable() ............................................. 521
sADMListEntry->IsEnabled() ....................................................... 521
sADMListEntry->IsEntryNameHidden() .......................................... 522
sADMListEntry->IsHierarchyExpanded() ......................................... 522
sADMListEntry->IsInBounds() ..................................................... 523
sADMListEntry->IsSelected() ..................................................... 523
sADMListEntry->IsSeparator() .................................................... 524
sADMListEntry->LocalToScreenPoint() ........................................... 524
sADMListEntry->LocalToScreenRect() ............................................ 525
sADMListEntry->MakeInBounds() ................................................ 525
sADMListEntry->MakeSeparator() ................................................ 526
sADMListEntry->ScreenToLocalPoint() .......................................... 526
sADMListEntry->ScreenToLocalRect() ............................................ 527
sADMListEntry->Select() ............................................................ 527
sADMListEntry->SendNotify() ..................................................... 528
ADMListEntry->SetBackgroundColor() ............................................ 528
ADMListEntry->SetDisabledPicture() ............................................ 529
ADMListEntry->SetDisabledPictureID() .......................................... 529
ADMListEntry->SetDividingLineColor() ......................................... 530
ADMListEntry->SetEntryTextRect() .............................................. 531
sAMLListEntry->SetFont() ........................................................ 531
sADMListEntry->SetID() ............................................................ 532
sADMListEntry->SetPicture() ..................................................... 533
sADMListEntry->SetPictureID() .................................................. 533
sADMListEntry->SetSelectedPicture() .......................................... 534
sADMListEntry->SetSelectedPictureID() ....................................... 534
sADMListEntry->SetText() .......................................................... 535
sAMLListEntry->SetTextColor() .................................................. 536
sADMListEntry->SetUserData() .................................................... 536
sADMListEntry->Update() .......................................................... 537
## ADM Help Support

- `sADMListEntry->GetHelpID()` .............................................. 537
- `sADMListEntry->Help()` .................................................. 538
- `sADMListEntry->SetHelpID()` ......................................... 538

## Chapter 17  The ADM Notifier Suite .................................... 541

- About the ADM Notifier Suite ............................................ 541
- Accessing the Suite ....................................................... 541
- ADM Notifier Functions ................................................ 541
- Using ADM Notifier Functions ....................................... 542
- ADM Notifier Types ....................................................... 542
- ADM Notifier Suite Functions .......................................... 547
  - `sADMNotifier->GetDialog()` ........................................... 547
  - `sADMNotifier->GetItem()` ............................................. 548
  - `sADMNotifier->GetNotifierType()` .................................. 548
  - `sADMNotifier->IsNotifierType()` ................................... 549
  - `sADMNotifier->SkipNextClipboardOperation()` .................... 549

## Chapter 18  The ADM Tracker Suite .................................... 551

- About the ADM Tracker Suite ............................................ 551
- Accessing the Suite ....................................................... 551
- ADM Trackers .............................................................. 551
- ADM Tracker Suite Functions ........................................... 552
  - `sADMTracker->Abort()` ............................................... 552
  - `sADMTracker->GetAction()` ......................................... 552
  - `sADMTracker->GetCharacter()` ...................................... 553
  - `sADMTracker->GetModifiers()` ..................................... 553
  - `sADMTracker->GetMouseState()` ................................... 553
  - `sADMTracker->GetPoint()` .......................................... 554
  - `sADMTracker->GetTime()` .......................................... 554
  - `sADMTracker->GetVirtualKey()` .................................... 555
  - `sADMTracker->ReleaseMouseCapture()` ............................... 555
  - `sADMTracker->TestAction()` ....................................... 556
  - `sADMTracker->TestModifier()` ..................................... 556

## Appendix A  ADM Folders and Files .................................... 559

- Files in SDK ADM Folders ................................................ 559

## Appendix B  ADM Glossary .................................................. 561
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 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;
ADMEnergySuite *sADMEnergy;
ADMHierarchyListSuite *sADMHierarchyList;
ADMIonSuite *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:

```c
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](http://partners.adobe.com).
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:

```c
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:

```c
SPAPI SPErrMsg 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:

```c
typedef struct SPMessageData {
    long SPCheck; /* kSPValidSPMessageData if a valid SPMessage */
    struct SPPPlugin *self; /* SPPPluginRef */
    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.

![ADM Objects](image)

**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:

```c
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`, `ADMTypes.h`, `ADMCustomResource.h`, and `ADMAgentTypes.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

```c
// 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;
    ASpanpoint 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.
#ifdef Platform_Carbon
typedef struct OpaqueGrafPtr* ASPortRef;
#else
typedef struct GrafPort* ASPortRef;
#endif

// ASWindowRef is the same as a Macintosh WindowPtr.
#ifdef 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;

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

// 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;

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


Events

There are five events received by all ADM user interface objects. These are listed and described in Table 1.1.

<table>
<thead>
<tr>
<th>Event</th>
<th>When Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>When object is created</td>
</tr>
<tr>
<td>Draw</td>
<td>When screen is invalidated or updated</td>
</tr>
<tr>
<td>Track</td>
<td>When mouse is over the object</td>
</tr>
<tr>
<td>Notify</td>
<td>When the object is hit</td>
</tr>
<tr>
<td>Destroy</td>
<td>When the object is disposed of</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Properties and Data</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Defines the general function of the object</td>
</tr>
</tbody>
</table>
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.

### Table 1.2 Basic Properties Common to All ADM UI Objects

<table>
<thead>
<tr>
<th>Properties and Data</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Style</td>
<td>Determines the appearance and/or behavior of the object</td>
</tr>
<tr>
<td>ID</td>
<td>Numeric reference to the object in its defining space (e.g., its resource number or item number)</td>
</tr>
<tr>
<td>Text</td>
<td>Depending on the item, it is usually the title, text value, or the name of an item</td>
</tr>
<tr>
<td>Visible</td>
<td>Whether or not the object is visible</td>
</tr>
<tr>
<td>Enabled</td>
<td>Whether or not the object is enabled</td>
</tr>
<tr>
<td>Active</td>
<td>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).</td>
</tr>
<tr>
<td>Known</td>
<td>Whether or not the object is known—an item is in a “known” state if it has a “good” or valid value.</td>
</tr>
<tr>
<td>Plug-in</td>
<td>A reference to the plug-in that created the object</td>
</tr>
<tr>
<td>UserData</td>
<td>A pointer to any special data assigned to the object when it was created</td>
</tr>
<tr>
<td>LocalRect</td>
<td>The size of the object (0,0)-based</td>
</tr>
<tr>
<td>BoundsRect</td>
<td>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.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Suite</th>
<th>Purpose</th>
<th>Associated Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic</td>
<td>Provides minimal dialog and resource functionality such as alerts, beeps, resource access, and string utilities.</td>
<td>ADMBasic.h</td>
</tr>
<tr>
<td>Dialog</td>
<td>ADM property access functions for dialog objects.</td>
<td>ADMDialog.h</td>
</tr>
<tr>
<td>Dialog Group</td>
<td>Functions for grouping dialogs into a docked palette.</td>
<td>ADMDialogGroup.h</td>
</tr>
<tr>
<td>Drawer</td>
<td>Functions for implementing custom drawer callbacks.</td>
<td>ADMDrawer.h</td>
</tr>
</tbody>
</table>
ADM Overview

ADM Object Specifics

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

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 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 1.4  ADM Dialog Object Types and Styles**

Modal Dialogs

- **kADMModalDialogStyle**
- **kADMAAlertDialogStyle**

Floating Dialogs

- **kADMTabbedFloatingDialogStyle**
- **kADMTabbedResizingFloatingDialogStyle**
- **kADMFloatingDialogStyle**
- **kADMResizingFloatingDialogStyle**
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.

```c
/* 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 and 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, sADMDial->Create() and sADMDial->Destroy(), are for non-modal (floating) dialogs. To make a non-modal dialog, the plug-in calls the sADMDial->Create() function. When the modeless dialog is no longer needed, the plug-in calls the sADMDial-
>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.
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

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 `kADMPicturePushButtonType` in Figure 1.7 is a default button.
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.

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.
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:

```c
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:

```c
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**

**kADMTextEditStaticMultilineType**
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
- kADMIinchUnits
- 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:

```c
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.
kADMTextEditPopupType
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.

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.

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

To get the name, you use the sADMEntry->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.

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:

```c
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.

![enabled and disabled scrollbar and slider](image)

**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.

**kADMLListBoxType**

**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.
ADM Overview
ADM Object Specifics

**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:

```c
/* 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:

```c
#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.

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

**kADMProgressBarType**

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

![ADM Progress Bar](image)

**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.

![ADM Chasing Arrows](image)

**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.

![ADM Dial](image)

**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:

```c
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 checkbox 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 typcast 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.

![ADM List and Entry Objects](image)

**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 `sADMEntry->SetText()` function to set the new entry’s text:

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

or more quickly by assigning it a menu resource ID:

```cpp
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 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.

![ADM Hierarchy List and List Entry Objects](image)

**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

<table>
<thead>
<tr>
<th>ADM Item Type</th>
<th>ADM Style</th>
<th>Windows Class Name</th>
<th>Window Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM Frame</td>
<td>kADMSunkenFrameStyle</td>
<td>“Button”</td>
<td>BS_GROUPBOX</td>
</tr>
<tr>
<td></td>
<td>kADMBlackFrameStyle</td>
<td>“Static”</td>
<td>SS_BLACKRECT</td>
</tr>
<tr>
<td></td>
<td>kADMGrayFrameStyle</td>
<td>“Static”</td>
<td>SS_GRAYRECT</td>
</tr>
<tr>
<td></td>
<td>kADMRaisedFrameStyle</td>
<td>“Static”</td>
<td>SS_WHITERECT</td>
</tr>
<tr>
<td></td>
<td>kADMBlackFrameStyle</td>
<td>“Static”</td>
<td>SS_BLACKFRAME</td>
</tr>
<tr>
<td></td>
<td>kADMGrayFrameStyle</td>
<td>“Static”</td>
<td>SS_GRAYFRAME</td>
</tr>
<tr>
<td></td>
<td>kADMRaisedFrameStyle</td>
<td>“Static”</td>
<td>SS_WHITEFRAME</td>
</tr>
<tr>
<td></td>
<td>kADMSunkenFrameStyle</td>
<td>“Static”</td>
<td>SS_ETCHEDHORZ</td>
</tr>
<tr>
<td></td>
<td>kADMSunkenFrameStyle</td>
<td>“Static”</td>
<td>SS_ETCHEDVERT</td>
</tr>
<tr>
<td></td>
<td>kADMSunkenFrameStyle</td>
<td>“Static”</td>
<td>SS_ETCHEDFRAME</td>
</tr>
<tr>
<td>ADMFrameStyle</td>
<td></td>
<td>“ADM Frame Type”</td>
<td></td>
</tr>
<tr>
<td>ADM List Box</td>
<td></td>
<td>“Listbox”</td>
<td></td>
</tr>
<tr>
<td>ADMListBoxStyle</td>
<td></td>
<td>“ADM List Box Type”</td>
<td></td>
</tr>
<tr>
<td>ADMListBoxStyle</td>
<td></td>
<td>“ADM Hierarchy List Box Type”</td>
<td></td>
</tr>
<tr>
<td>ADM Picture Push Button (0)</td>
<td>item name == MAKEINTRESOURCE(pictureID)</td>
<td>“ADM Picture Push Button Type”</td>
<td>0</td>
</tr>
<tr>
<td>ADM Picture Radio Button (0)</td>
<td>item name == MAKEINTRESOURCE(pictureID)</td>
<td>“ADM Picture Radio Button Type”</td>
<td>0</td>
</tr>
<tr>
<td>ADM Picture Static (0)</td>
<td>item name == MAKEINTRESOURCE(pictureID)</td>
<td>“Static”</td>
<td>SS_Bitmap</td>
</tr>
</tbody>
</table>
### Table 1.4  Windows ADM Items

<table>
<thead>
<tr>
<th>ADM Item Type</th>
<th>ADM Style</th>
<th>Windows Class Name</th>
<th>Window Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>item name == MAKEINTRESOURCE(pictureID)</td>
<td>“Static”</td>
<td>SS_Icon</td>
<td></td>
</tr>
<tr>
<td>item name == MAKEINTRESOURCE(pictureID)</td>
<td>“Static”</td>
<td>SS_ENHMETAFILE</td>
<td></td>
</tr>
<tr>
<td>item name == MAKEINTRESOURCE(pictureID)</td>
<td>“ADM Picture Static Type”</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADM Popup List</td>
<td>“Combobox”</td>
<td>CBS_DROPDOWNLIST</td>
<td></td>
</tr>
<tr>
<td>ADM Popup Control</td>
<td>“ADM Popup List Type”</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADM Popup Control Button</td>
<td>“ADM Popup Control Button Type”</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADM Popup Spin Edit Control</td>
<td>“ADM Popup Spin Edit Control Type”</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADM Popup Menu</td>
<td>ADMPopupMenuStyle</td>
<td>“ADM Popup Menu Type”</td>
<td>0</td>
</tr>
<tr>
<td>ADM Resize</td>
<td>“ADM Resize Type”</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADM Scrollbar</td>
<td>“Scrollbar”</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADM Scrolling Popup List</td>
<td>“Combobox”</td>
<td>CBS_DROPDOWNLIST</td>
<td>WS_VSCROLL</td>
</tr>
<tr>
<td>ADM Scrolling Popup List</td>
<td>“ADM Scrolling Popup List Type”</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADM Slider</td>
<td>ADMSliderStyle</td>
<td>“MSCtls_Trackbar32”</td>
<td>0</td>
</tr>
<tr>
<td>ADM Spin Edit</td>
<td>ADMSpinEditStyle</td>
<td>“ADM Spin Edit Type”</td>
<td></td>
</tr>
<tr>
<td>ADM Spin Edit Popup</td>
<td>kADMSingleLineEditPopupStyle</td>
<td>“Combobox”</td>
<td>CBS_DROPDOWN</td>
</tr>
<tr>
<td>ADM Spin Edit Scrolling Popup</td>
<td>kADMSingleLineEditPopupStyle</td>
<td>“Combobox”</td>
<td>CBS_DROPDOWNLIST</td>
</tr>
</tbody>
</table>
### TABLE 1.4 Windows ADM Items

<table>
<thead>
<tr>
<th>ADM Item Type</th>
<th>ADM Style</th>
<th>Windows Class Name</th>
<th>Window Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM Spin Edit</td>
<td>ADMSpinEditPopupStyle</td>
<td>“ADM Spin Edit</td>
<td>“ADM Spin Edit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scrolling Popup</td>
<td>Scrolling Popup Type”</td>
</tr>
<tr>
<td></td>
<td>ADM Text Check Box</td>
<td>“Button”</td>
<td>BS_CHECKBOX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Button”</td>
<td>BS_AUTOCHECKBOX</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Button”</td>
<td>BS_3STATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Button”</td>
<td>BS_AUTO3STATE</td>
</tr>
<tr>
<td></td>
<td>“ADM Text Check Box Type”</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ADM Text Edit</td>
<td>kADMLeftJustify</td>
<td>“Edit”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Edit”</td>
<td>ES_LEFT</td>
</tr>
<tr>
<td></td>
<td>kADMCenterJustify</td>
<td>“Edit”</td>
<td>ES_CENTER</td>
</tr>
<tr>
<td></td>
<td>kADMRightJustify</td>
<td>“Edit”</td>
<td>ES_RIGHT</td>
</tr>
<tr>
<td></td>
<td>kADMNumericEditStyle</td>
<td>“Edit”</td>
<td>ES_NUMBER</td>
</tr>
<tr>
<td></td>
<td>(Auto sets if you call</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>SetXValue()</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADM TextEditStyle</td>
<td>“ADM Text Edit Type”</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADM Password Text Edit</td>
<td>“ADM Text Edit Type”</td>
<td>ES_PASSWORD</td>
</tr>
<tr>
<td></td>
<td>ADM Text Edit Scrolling Popup</td>
<td>kADMSingleLineEditPopupStyle</td>
<td>“Combobox”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CBS_DROPDOWN</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>WS_VSCROLL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADM TextEditPopupStyle</td>
<td>“ADM Text Edit</td>
<td>“ADM Text Edit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scrolling Popup</td>
<td>Scrolling Popup Type”</td>
</tr>
<tr>
<td></td>
<td>ADM Text Edit Multi Line</td>
<td>kADMNumericEditStyle</td>
<td>“Edit”</td>
</tr>
<tr>
<td></td>
<td>(Auto sets if you call</td>
<td>(“Edit”</td>
<td>ES_MULTILINE</td>
</tr>
<tr>
<td></td>
<td><code>SetXValue()</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADM Text Edit Multi Line Type</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>ADM Text Edit Popup</td>
<td>kADMSingleLineEditPopupStyle</td>
<td>“Combobox”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CBS_DROPDOWN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ADM Text Edit Popup</td>
<td>ADMTextEditPopupStyle</td>
<td>“ADM Text Edit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“ADM Text Edit</td>
<td>“ADM Text Edit Popup</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Popup Type”</td>
<td>Type”</td>
</tr>
<tr>
<td></td>
<td>ADM Text Push Button</td>
<td>Default</td>
<td>“Button”</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Button”</td>
<td>BS_DEF_PUSHBUTTON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Button”</td>
<td>BS_PUSHBUTTON</td>
</tr>
</tbody>
</table>
### Table 1.4 Windows ADM Items

<table>
<thead>
<tr>
<th>ADM Item Type</th>
<th>ADM Style</th>
<th>Windows Class Name</th>
<th>Window Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Button”</td>
<td>BS_USERBUTTON</td>
<td>“Button”</td>
<td>BS_OWNERDRAW</td>
</tr>
<tr>
<td>“Button”</td>
<td>BS_OWNERDRAW</td>
<td>“ADM Text Push Button Type”</td>
<td>0</td>
</tr>
<tr>
<td>ADM Text Radio Button</td>
<td>ADMRadioButtonStyle</td>
<td>“Button”</td>
<td>BS_RADIOBUTTON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“Button”</td>
<td>BS_AUTORADIOBUTTON</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“ADM Text Radio Button Type”</td>
<td>0</td>
</tr>
<tr>
<td>ADM Text Static</td>
<td>kADMLeftJustify</td>
<td>“Static”</td>
<td>SS_LEFT</td>
</tr>
<tr>
<td></td>
<td>kADMCenterJustify</td>
<td>“Static”</td>
<td>SS_CENTER</td>
</tr>
<tr>
<td></td>
<td>kADMRightJustify</td>
<td>“Static”</td>
<td>SS_RIGHT</td>
</tr>
<tr>
<td></td>
<td>kADMLeftJustify</td>
<td>“Static”</td>
<td>SS_LEFTNOWORDWRAP</td>
</tr>
<tr>
<td></td>
<td>kADMLeftJustify</td>
<td>“Static”</td>
<td>SS_SIMPLE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“ADM Text Static Type”</td>
<td>0</td>
</tr>
<tr>
<td>ADM Text Static Multi Line</td>
<td></td>
<td>“Edit”</td>
<td>ES_READONLY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>“ADM Text Static Multi Line Type”</td>
<td>0</td>
</tr>
<tr>
<td>ADM User</td>
<td>“ADM User Type”</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADM Dial</td>
<td>“ADM Dial Type”</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ADM Tabbed Menu</td>
<td>Deprecated in V2.8—do not use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Custom</td>
<td>&lt;custom item name&gt;</td>
<td>“”</td>
<td></td>
</tr>
<tr>
<td>ADM Item Type</td>
<td>Mac Dialog Item</td>
<td>CDEF Res ID</td>
<td>Variation</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------</td>
<td>------------</td>
<td>-----------</td>
</tr>
<tr>
<td>ADM Frame</td>
<td>User Item</td>
<td>1000</td>
<td>ADMFrameStyle</td>
</tr>
<tr>
<td>ADM List Box</td>
<td>Control Item</td>
<td>1010</td>
<td>ADMLListBoxStyle</td>
</tr>
<tr>
<td>ADM Picture Check Box</td>
<td>Control Item</td>
<td>1023</td>
<td>ADMPictureButtonStyle</td>
</tr>
<tr>
<td>ADM Picture Push Button</td>
<td>Control Item</td>
<td>1020</td>
<td>ADMPictureButtonStyle</td>
</tr>
<tr>
<td>ADM Picture Radio Button</td>
<td>Control Item</td>
<td>1021</td>
<td>ADMPictureButtonStyle</td>
</tr>
<tr>
<td>ADM Picture Static</td>
<td>Control Item</td>
<td>1022</td>
<td>0</td>
</tr>
<tr>
<td>ADM Popup List</td>
<td>Control Item</td>
<td>63</td>
<td>0</td>
</tr>
<tr>
<td>ADM Popup Control</td>
<td>Control Item</td>
<td>1055</td>
<td>0</td>
</tr>
<tr>
<td>ADM Popup Control Button</td>
<td>Control Item</td>
<td>1056</td>
<td>0</td>
</tr>
<tr>
<td>ADM Popup Spin Edit Control</td>
<td>Control Item</td>
<td>1057</td>
<td>0</td>
</tr>
</tbody>
</table>
### Table 1.5 Macintosh ADM Items

<table>
<thead>
<tr>
<th>ADM Item Type</th>
<th>Mac Dialog Item</th>
<th>CDEF Res ID</th>
<th>Variation</th>
<th>Value</th>
<th>Min</th>
<th>Max</th>
<th>Other</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM Popup Menu</td>
<td>Control Item</td>
<td>1030</td>
<td>ADMPopupMenuStyle</td>
<td>MenuID = MenuResID (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Resize</td>
<td>Control Item</td>
<td>1040</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Scrollbar</td>
<td>Control Item</td>
<td>1</td>
<td>0</td>
<td>IntValue (1)</td>
<td>IntMin (1)</td>
<td>IntMax (1)</td>
<td></td>
</tr>
<tr>
<td>ADM Scrolling Popup List</td>
<td>Control Item</td>
<td>1031</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Slider</td>
<td>Control Item</td>
<td>1050</td>
<td>0</td>
<td>IntValue (1)</td>
<td>IntMin (1)</td>
<td>IntMax (1)</td>
<td></td>
</tr>
<tr>
<td>ADM Spin Edit</td>
<td>Control Item</td>
<td>1060</td>
<td>ADMSpinEditStyle</td>
<td>ADMJustify (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Spin Edit Popup</td>
<td>Control Item</td>
<td>1061</td>
<td>ADMSpinEditPopupStyle</td>
<td>ADMJustify (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Spin Edit Scrolling Popup</td>
<td>Control Item</td>
<td>1062</td>
<td>ADMSpinEditPopupStyle</td>
<td>ADMJustify (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Text Check Box</td>
<td>Check Box Item</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Text Edit</td>
<td>Edit Text Item</td>
<td>1070</td>
<td>ADMTextEditStyle</td>
<td>ADMJustify (2)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Text Edit Multi Line</td>
<td>Control Item</td>
<td>1073</td>
<td>0</td>
<td>ADMJustify (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Text Edit Popup</td>
<td>Control Item</td>
<td>1071</td>
<td>ADMTextEditPopupStyle</td>
<td>ADMJustify (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM Text Edit Scrolling Popup</td>
<td>Control Item</td>
<td>1075</td>
<td>ADMTextEditPopupStyle</td>
<td>ADMJustify (1)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adobe Dialog Manager Programmer’s Guide and Reference
**Table 1.5  Macintosh ADM Items**

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

```c
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.
ADM Overview
Event Callbacks

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:

```c
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:

**TABLE 1.6  ADM Object Customization**

<table>
<thead>
<tr>
<th>Event/Customization</th>
<th>ADM Dialog</th>
<th>ADM Item</th>
<th>ADM Entry¹</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customizable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Can Call Default</td>
<td>Y</td>
<td>Y</td>
<td>N²</td>
</tr>
<tr>
<td>Draw</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customizable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Can Call Default</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Track</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customizable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Can Call Default</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Notify</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customizable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Can Call Default</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Destroy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Customizable</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Can Call Default</td>
<td>Y</td>
<td>Y</td>
<td>N²</td>
</tr>
</tbody>
</table>

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.
ADM Overview

Event Callbacks

1

Adobe Dialog Manager Programmer's Guide and Reference

```c
ASErr 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);
}

ASErr myListEntryInit(ADMEntryRef 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:

```c
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:

```c
ADMEntryRef 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:

```c
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 ADMDrawer 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:

```c
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:

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

It would be used something like this:

```c
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:

```c
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.

```c
#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:

```c
ASErr 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:

```c
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:

```c
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:

```c
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, \texttt{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 \texttt{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**

<table>
<thead>
<tr>
<th>Windows Class Name</th>
<th>Window Style</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;custom item name&gt;</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

**Table 1.8  Macintosh Custom ADM Item Resource Information**

<table>
<thead>
<tr>
<th>Mac Dialog Item</th>
<th>CDEF Res ID</th>
<th>Control Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control Item</td>
<td>1090</td>
<td>&lt;custom item name&gt;</td>
</tr>
</tbody>
</table>

The custom item can also be created at runtime with the \texttt{sADMDialog->CreateItem()} function:

\begin{verbatim}
    sADMDialog->CreateItem(myDialog, kMyCustomItemID, 
                         <custom item name>, &myCustomItemRect);
\end{verbatim}

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 \texttt{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 \texttt{IADM (Interface to ADM)} directory in several Adobe SDKs.
The following example shows the same process in both styles:

```c
// 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:

```c
class IADMDialog
class IADMItem
class IADMList
class IADMEntry

class IADMNNotifier
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:

```cpp
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:

```cpp
ADMBasicSuite *sADMBasic;
ADMDialogSuite *sADMDialog;
ADMDialogGroupSuite *sADMDialogGroup;
ADMDrawerSuite *sADMDrawer;
ADMEntrySuite *sADMEntry;
ADMHierarchyListSuite *sADMHierarchyList;
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](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.
Introduction

Acrobat 5.0 and above 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 and snippets in the Acrobat SDK illustrate the use of ADM.

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:

```c
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 81 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,
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:

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

Then set the value depending on the platform.

```c
#ifdef WIN_PLATFORM
    retVal = AVSweetPeaSetResourceAccess(sADMPluginRef,
                                        (SPPlatformAccessRef)gHINSTANCE);
#endif
```
#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)&macRecInfo);
#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 and above 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 and plists**

To use ADM with Acrobat, you use either a ‘carb’ resource or a plist. The 6.0 plug-ins use *plist*, while the 5.x plug-ins use ‘carb’.

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

Also available from Apple:
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:

```plaintext
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
  PUSHDOWNLIST  "&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
  PUSHDOWNLIST  "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
#endif
```
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 is 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},
    movableDBoxProc,
    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:
    },
Using ADM with Adobe Photoshop

Platform-Specific Resources

/* [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
},
{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"
  }
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>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sSPBasic</td>
<td>A pointer to the <code>SPBasicSuite</code> struct. This struct contains two host member functions: <code>AcquireSuite()</code> and <code>ReleaseSuite()</code>, which are invoked by <code>PIUSuitesAcquire()</code> to acquire the suites.</td>
</tr>
</tbody>
</table>
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.

```c
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,
kJTop_staticText,
kJTop_editText,
kJLeft_staticText,
kJLeft_editText,
kJBottom_staticText,
kJBottom_editText,
kJRight_staticText,
kJRight_editText

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

\textbf{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 \texttt{DoUI()} function. Refer to the \texttt{sADMDialog-&gt;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 \texttt{DoUIInit()}, and follows the calling conventions for the \texttt{ADMDialogInitProc} callback as documented in the \texttt{sADMDialog-&gt;Modal()} function. The initialization function initializes each item in the dialog, provides focus to the first text field (\textit{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 \texttt{Cancel} button and the pop-up menu is extensive, separate functions are used to accomplish these tasks.

\section*{Operation}

The \texttt{DoUI()} function handles interaction with the UI items that make up the dialog. If the user presses the \texttt{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.

\begin{verbatim}
SPErr DoUI()
{
  SPErr error = noErr;

  int item = kDNoUI; // Error value.

  SaveParameters(); // Save our parameters, just in case.
  if (sADMDialog != NULL)
  {
    item = sADMDialog-&gt;Modal(
       gPlugInRef,


dj
```
"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.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pluginRef</td>
<td>gPlugInRef</td>
<td>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 gPlugInRef is passed in by the host application via PluginMain(), which is called when the plug-in is first loaded.</td>
</tr>
<tr>
<td>name</td>
<td>FrameSelection</td>
<td>The name of the new dialog. In this case we choose FrameSelection.</td>
</tr>
<tr>
<td>dialogID</td>
<td>kDialogID</td>
<td>A dialog ID (uiID) is provided. It is the same ID as used in the FrameSelect.r 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 PIUtilities.r for those define statements.)</td>
</tr>
</tbody>
</table>
Using ADM with Adobe Photoshop  
Building, Presenting, and Using the Dialog

The first function that is executed when \texttt{DoUI()} is run is \texttt{DoUIInit()}:

```c
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);
}
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style</td>
<td>kADMModalDialogStyle</td>
<td>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.</td>
</tr>
<tr>
<td>initProc</td>
<td>DoUIInit</td>
<td>The name of the ADMDialogInitProc callback function that initializes the dialog elements using ADM routines. In this case, it is the DoUIInit() function.</td>
</tr>
<tr>
<td>data</td>
<td>NULL</td>
<td>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.</td>
</tr>
<tr>
<td>inOptions</td>
<td>0</td>
<td>Not used here. See sADMDialog-&gt;Modal().</td>
</tr>
</tbody>
</table>
// Hilight 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, ADMEntry, 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.

```c
static SPErr InitCancelButton(ADMDialogRef dialog)
{
    SPErr 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,
```
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:

```c
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:

```c
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.

```c
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:

```c
static void SetTextToReset
{
    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:

```c
ADMActionMask mask = sADMItem->GetMask(item);
    sADMItem->SetMask
    (item,
      mask | kADMModKeyDownMask | kADMModKeyUpMask | kADMLeaveMask | 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:

```c
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:

```c
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:

```c
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.
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](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 `sADMDialoRef Create()` function as shown in the code below.

```c
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.

```c
// 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;

    return error;
}
```

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

```c
// ADM Suites
#include "ADMBasic.h"
#include "ADMDialog.h"
#include "ADMItem.h"
#include "ADMIcon.h"
#include "ADMList.h"
#include "ADMDialogGroup.h"
#include "ADMNotifier.h"
#include "ADMEntity.h"
#include "ADMTacker.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:

```c
#define kADMNonModalDialogID 16000
#define kpopupItem 5
#define kpopupMenuID 800

#define kbeepItem 3
#define kbeepBeepItem 4

#define kspinEditPopupItem 6
#define kspinEditPopupMenuID 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`, `ADMEntryRef`, etc.

```c
ASErr ASAPI DlgInit(ADMDialogRef dlg)
{
    PUSHGLOBALS
    ASErr fxErr = kNoErr;
    AIAppContextHandle AppContext;
    ADMItemRef menuItemRef, popupItemRef,
    beepButtonItemRef,
    beepBeepButtonItemRef,
    spinEditPopupItemRef,
    spinEditPopupMenuRef,
    spinEditPopupListRef;
    ADMListRef layerListRef = 0;
    ADMListRef menuListRef, popupListRef,
    spinEditPopupListRef;
    ADMEntryRef 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.

![An ADM Popup List](image)

**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:

```cpp
// 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
    sADMEntry->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(spinEditPopupListRef);
sADMEntry->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->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);
}
Using ADM with Adobe Illustrator

A Modeless Dialog Example Using Illustrator

```c
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.

```c
// 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);
```

```c
// 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,
ADMNNotifierRef 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;
    ADMEntryRef 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 = sADMEntry->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.

![Flyout Popup Menu with Option Key Pressed](image)

**FIGURE 4.5   Flyout Popup Menu with Option Key Pressed**

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

```c
/* 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)
{
    PUSHGLOBALS
    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,
   kADMModKeyDownModifier);

// checks if the (mac)COMMAND (win)CONTROL modifier is pressed
commandControlDown = sADMTracker->TestModifier(tracker,
   kADMMenuKeyDownModifier);

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

if (menuListRef)
{
   // if command is pressed, enable the first menu item,
   // otherwise disable
   sADMEntry->Enable(sADMList->IndexEntry(menuListRef, 0), commandOptionDown);
   // if command is pressed, enable the second menu item,
   // otherwise disable
   sADMEntry->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.
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:

```c
ADMDialogSuite *sADMDiallog;
error = sSPBasic->AcquireSuite(ADMDialogSuite, ADMDialogSuiteVersion2,
&*sADMDiallog);
if (error) goto ... //handle error
```

the *sADMDiallog 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_diallogP` instead of `sADMDiallog`), 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.
Using ADM with Adobe After Effects

Easy_Cheese Plug-in
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:

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

with the version constants:

```c
#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:

```c
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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inPluginRef</code></td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td><code>inStringID</code></td>
<td>The resource ID.</td>
</tr>
<tr>
<td><code>inStringIndex</code></td>
<td>1-based index of the string in the list.</td>
</tr>
<tr>
<td><code>outString</code></td>
<td>The returned string.</td>
</tr>
<tr>
<td><code>inMaxLen</code></td>
<td>Maximum number of characters to be read.</td>
</tr>
</tbody>
</table>

**Returns**

The number of characters actually read.

**Example**

```c
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 (CURSOR).

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPluginRef</td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td>inCursorID</td>
<td>Cursor resource ID.</td>
</tr>
</tbody>
</table>

Returns

None.

User Interface Functions

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

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 sADMDIal->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**

<table>
<thead>
<tr>
<th>inPlugin</th>
<th>Plug-in reference.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inText1</td>
<td>First user-defined line of text.</td>
</tr>
<tr>
<td>inText2</td>
<td>Second user-defined line of text.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMBasic->PluginAboutBox()`

---

**sADMBasic->AreToolTipsEnabled()**

Determine whether ADM tool tips are on or off.

\[
\text{ASBoolean ASAPI \(*\text{AreToolTipsEnabled}\) () ;}
\]

**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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>Reference to an instance of a dialog.</td>
</tr>
</tbody>
</table>

**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

```c
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inWhere</td>
<td>The position argument; determines the location of the dialog. Type: ASPoint (see ASTypes.h)</td>
</tr>
<tr>
<td>inColor</td>
<td>Allows for setting of initial &quot;picked&quot; 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 ASTypes.h)</td>
</tr>
<tr>
<td>outColor</td>
<td>Returns the selected color. Type: ASRGBColor (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEnable</td>
<td>Boolean indicating whether tool tips should be turned on (true) or off (false).</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMBasic->HideToolTip()
sADMBasic->ShowToolTip()
The ADM Basic Suite
User Interface Functions

sADMBasic->AreToolTipsEnabled()
sADMBasic->AreToolTipsSticky()

**Description**

Present an error dialog to the user

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

**Parameters**

- **inErrorString**
  Error string text.

**Returns**

None.

**See also**

- sADMBasic->MessageAlert()
- sADMBasic->LightweightErrorAlert()
- sADMBasic->QuestionAlert()
- sADMBasic->SetAlertButtonText()
- sADMBasic->YesNoAlert()

**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.

**See also**

- sADMBasic->MessageAlert()
- sADMBasic->LightweightErrorAlert()
- sADMBasic->QuestionAlert()
- sADMBasic->SetAlertButtonText()
- sADMBasic->YesNoAlert()

**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.
sADMBasic->HideToolTip()

void ASAPI (*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()

void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inTitle</td>
<td>Text displayed in the title bar of the dialog.</td>
</tr>
<tr>
<td>inText</td>
<td>Text displayed in the dialog.</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

sADMBasic->AboutBox()

### sADMBasic->QuestionAlert()

Present a question dialog to the user

```c
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`:

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inQuestionString</td>
<td>Alert string text.</td>
</tr>
</tbody>
</table>

### Returns

Enumerated value of type `ADMAnswer`. 
sADMBasic->SetAlertButtonText()  
Set the button text for an Alert dialog

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inLeftChoice</code></td>
<td>Text for left button. Default: Yes.</td>
</tr>
<tr>
<td><code>inMiddleChoice</code></td>
<td>Text for middle button. Default: No.</td>
</tr>
<tr>
<td><code>inRightChoice</code></td>
<td>Text for right button. Default: Cancel.</td>
</tr>
</tbody>
</table>

**Returns**

See `sADMBasic->QuestionAlert()` or `sADMBasic->YesNoAlert()`.

**See also**

`sADMBasic->QuestionAlert()`  
`sADMBasic->YesNoAlert()`

sADMBasic->ShowToolTip()  
Show ADM tool tips

```c
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.
The ADM Basic Suite
User Interface Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inWhere</code></td>
<td>Position where the tool tip is to be displayed. Type: <code>ASPoint</code> (see <code>ASTypes.h</code>)</td>
</tr>
<tr>
<td><code>inTip</code></td>
<td>Tool tip text.</td>
</tr>
</tbody>
</table>

See also

- `sADMBasic->EnableToolTips()`
- `sADMBasic->HideToolTip()`
- `sADMBasic->AreToolTipsEnabled()`
- `sADMBasic->AreToolTipsSticky()`

---

`sADMBasic->StandardGetFileDialog()`  
Open standard file open dialog

```c
```

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:

```c
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:

```c
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:

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inMessage</td>
<td>Text displayed at the top of the dialog.</td>
</tr>
<tr>
<td>inFilter</td>
<td>Filter that determines what information is displayed.</td>
</tr>
<tr>
<td>inStartingDir</td>
<td>Starting directory.</td>
</tr>
<tr>
<td>inStartingFile</td>
<td>Starting file.</td>
</tr>
<tr>
<td>outResult</td>
<td>File selected by user.</td>
</tr>
</tbody>
</table>

Returns

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

See also

`sADMBasic->StandardPutFileDialog()`
`sADMBasic->StandardGetDirectoryDialog()`

---

`sADMBasic->StandardGetDirectoryDialog()`  
Open standard directory select dialog

```c
```

Description

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

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inMessage</td>
<td>Text displayed at the top of the dialog.</td>
</tr>
<tr>
<td>inStartingDir</td>
<td>Starting directory.</td>
</tr>
<tr>
<td>outResult</td>
<td>Directory selected by user.</td>
</tr>
</tbody>
</table>
The ADM Basic Suite
User Interface Functions

Returns

true if directory was found and returned; false otherwise.

See also

sADMBasic->StandardGetFileDialog()
sADMBasic->StandardPutFileDialog()

sADMBasic->StandardPutFileDialog()
Open standard file save dialog


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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inMessage</td>
<td>Text displayed at the top of the dialog.</td>
</tr>
<tr>
<td>inFilter</td>
<td>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: ADMPlatformFileTypesSpecification3 (see ADMBasic.h)</td>
</tr>
<tr>
<td>inStartingDir</td>
<td>Starting directory. Type: SPPlatformFileSpecification (see ADMBasic.h)</td>
</tr>
<tr>
<td>inStartingFile</td>
<td>Starting file.</td>
</tr>
<tr>
<td>outResult</td>
<td>File selected by user. Type: SPPlatformFileSpecification (see ADMBasic.h)</td>
</tr>
</tbody>
</table>

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`:

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inADMColor</td>
<td>ADM color for conversion. Type: ADMColor (see ADMTypes.h)</td>
</tr>
<tr>
<td>outRGBColor</td>
<td>ADM color converted to RGB value. Type: ASRGBColor (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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 ASAPI (*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 ADMTypes.h). |

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 ASAPI (*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) |
The ADM Basic Suite
Utility Functions

Returns
None.

See also
sADMBasic->SetDefaultIncrements()

sADMBasic->GetLastADMError()
Get the last ADM error data

ASBoolean ASAPI (*GetLastADMError)(ASInt32* outError, ASInt32* outErrorData);

Description
The GetLastADMError() function is used to retrieve relevant information about why a dialog failed to load after using sADMDialog->Create(), sADMDialog->Modal(), or sADMDialog->CreateGroupInDialog().

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outError</td>
<td>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 ADMCustomResource.h and ADMDialogGroup.h.</td>
</tr>
<tr>
<td>outErrorData</td>
<td>Data associated with outError (e.g., dialog reference).</td>
</tr>
</tbody>
</table>

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()

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 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()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPoint</td>
<td>A point on the screen. Type: ASPoint (see ASTypes.h)</td>
</tr>
<tr>
<td>outDimensions</td>
<td>Screen dimensions. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outDimensions</td>
<td>Workspace dimensions. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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.
The ADM Basic Suite
Utility Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPluginRef</td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td>inString</td>
<td>ZString.</td>
</tr>
<tr>
<td>outString</td>
<td>Translated ZString.</td>
</tr>
<tr>
<td>ioBufferSize</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inFPS</td>
<td>FPS value.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inAppUnits</code></td>
<td>Unit preference. One of 10 enumerated values of type <code>ADMUnits</code> (see <code>ADMTypes.h</code>).</td>
</tr>
<tr>
<td><code>inActualUnits</code></td>
<td>Units for the <code>inAppUnits</code> unit preference. Type: <code>ADMUnits</code> (see <code>ADMTypes.h</code>).</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

`sADMBasic->GetAppUnits()`

---

### `sADMBasic->SetDefaultIncrements()`

Sets the defaults for spinner, slider, and scroll bar cursor movement.

```c
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

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

### Returns

None.
The ADM Basic Suite

Utility Functions

See also

sADMBasic->GetDefaultIncrements()

sADMBasic->SetNumbersArePoints()  Sets the unit for numbers as points

void ASAPI (*SetNumbersArePoints)(ASBoolean inPoints);

Description

The GetNumbersArePoints() 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPoints</td>
<td>If true, points will be the default unit of measurement in the application. If false, they will not be.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inText</td>
<td>Floating point value to convert to a C string.</td>
</tr>
<tr>
<td>outValue</td>
<td>inValue converted to a C string.</td>
</tr>
<tr>
<td>inUnits</td>
<td>Used to scale inValue after conversion. Type: ADMUnits</td>
</tr>
</tbody>
</table>

(see ADMTypes.h)

Returns

true if the conversion was successful; false otherwise.
See also

sADMBasic->ValueToString()

---

**sADMBasic->ValueToString()**

Convert a value to a string

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inValue</code></td>
<td>Floating point value to convert to a C string.</td>
</tr>
<tr>
<td><code>outText</code></td>
<td><code>inValue</code> converted to a C string.</td>
</tr>
<tr>
<td><code>inMaxLen</code></td>
<td>Maximum length for string <code>outText</code>.</td>
</tr>
<tr>
<td><code>inUnits</code></td>
<td>Used to scale <code>inValue</code> before conversion. Type: <code>ADMUnits</code> (see <code>ADMTypes.h</code>)</td>
</tr>
<tr>
<td><code>inPrecision</code></td>
<td>Maximum number of decimal places that are used in the final string.</td>
</tr>
<tr>
<td><code>inAlwaysAppendUnits</code></td>
<td>If <code>true</code>, units are appended to string.</td>
</tr>
</tbody>
</table>

**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

```c
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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inMenu</code></td>
<td>Contextual menu to display.</td>
</tr>
<tr>
<td><code>inDialog</code></td>
<td>Dialog over which to display <code>inMenu</code></td>
</tr>
<tr>
<td><code>inDialogPoint</code></td>
<td>Point within <code>inDialog</code> at which to display <code>inMenu</code>. Type: <code>ASPoint</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**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()`
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:

```c
#define kADMDialogSuite "ADM Dialog Suite"
```

with the version constant:

```c
#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:

```c
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:

```c
typedef enum
{
    kADMModalDialogStyle = 0,
    kADMAlertDialogStyle = 1,
    kADMFloatingDialogStyle = 2,
    kADMTabbedFloatingDialogStyle = 3,
    kADMResizingFloatingDialogStyle = 4,
    kADMTabbedResizingFloatingDialogStyle = 5,
    kADMPopupDialogStyle = 6,
    kADMNoCloseFloatingDialogStyle = 7,
    kADMSSystemAlertDialogStyle = 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:

```c
typedef enum
{
    kADMModalDialogStyle = 0,
    kADMAlertDialogStyle = 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:

```c
typedef enum
{
    kADMUniqueItemID = 0,
    kADMFirstItemID = -1,
    kADMLastItemID = -2,
    kADMDefaultItemID = -3,
    kADMCancelItemID = -4,
    kADMMenuItemID = -5,
    kADMDockItemID = -6,
    kADMResourceUniqueItemID = -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.

```c
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

**sADMDial->AbortTimer()**

Abort a timer

```c
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 `ADMTypes.h`)

**Returns**

None.
See also

sADMDialog->CreateTimer()

---

### sADMDialog->Activate()

**Make a dialog active or inactive**

```c
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**

```c
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.
The ADM Dialog Suite

ADM Dialog Suite Functions

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inPluginRef</code></td>
<td>Plug-in reference.</td>
</tr>
</tbody>
</table>
### 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. 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.

#### TABLE 7.1 Possible values for inOptions

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>kADMTabbedDialogShowsCycleOption (1L &lt;&lt; 0)</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>kADMPassMouseDownEventInModalDialogOption (1L &lt;&lt; 1)</code></td>
<td>Used to allow modal dialogs to pass mouse down events through to the user dialog tracker.</td>
</tr>
<tr>
<td><code>kReservedForUseByCustomResourcesOption (1L &lt;&lt; 2)</code></td>
<td>Reserved.</td>
</tr>
<tr>
<td><code>kADMKeypadEnterNotDefault (1L &lt;&lt; 3)</code></td>
<td><code>enter</code> key does not activate default item.</td>
</tr>
</tbody>
</table>
The ADM Dialog Suite

ADM Dialog Suite Functions

TABLE 7.1 Possible values for inOptions

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>kADMCreateDialogItems</td>
<td>Reduces flicker by creating items hidden.</td>
</tr>
<tr>
<td>HiddenByDefault (1L &lt;&lt; 4)</td>
<td>Forces for all items within dialog, except as overridden.</td>
</tr>
<tr>
<td>kADMForceRomanFontForDialog (1L &lt;&lt; 5)</td>
<td>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.)</td>
</tr>
<tr>
<td>kADMTrackEnterBeforeDialogDoesOK (1L &lt;&lt; 6)</td>
<td>Helps make popup dialogs in palettes layer properly.</td>
</tr>
<tr>
<td>kADMModalDialogHasPaletteSystemControlsOption (1L &lt;&lt; 7)</td>
<td>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.</td>
</tr>
<tr>
<td>kADMCreatePopupMenuDialogAsFloatingClassOption (1L &lt;&lt; 8)</td>
<td>Reserved.</td>
</tr>
<tr>
<td>1L &lt;&lt; 28 and higher shifts</td>
<td>Reserved.</td>
</tr>
</tbody>
</table>

Returns

An ADM Dialog.

See also

- sADMDialog->Destroy()
- sADMDialog->Modal()
- sADMDialog->GetUserData()
Example

```c
sADMDialog->Create(message->d.self, "ADMNonModalDialog",
    kADMDiNonModalDialogID, 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);
    sADMIItem->SetNotifyProc(dlgItem, DlgOKButtonClickProc);

    // 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);
    sADMIItem->SetBooleanValue(dlgItem, true);

    return kNoErr;
}
```

**sADMDialog->CreateGroupInDialog()**

Create a group in a dialog

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inDialog</strong></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><strong>inPluginRef</strong></td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td><strong>inName</strong></td>
<td>Internal identifier for the dialog. Must be unique. The suggested convention</td>
</tr>
<tr>
<td></td>
<td>is to use a leading prefix of your company's name, as in <strong>MyCompanyDialogxxxx</strong>.</td>
</tr>
<tr>
<td><strong>inDialogID</strong></td>
<td>Platform native resource ID. This ID refers to the platform-specific code</td>
</tr>
<tr>
<td></td>
<td>provided to ADM to create the dialog on either the Macintosh or PC.</td>
</tr>
</tbody>
</table>
## 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 item 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inItemID</code></td>
<td>Index of the item within <code>inDialog</code>'s list of items.</td>
</tr>
<tr>
<td><code>inItemType</code></td>
<td>Type of item to be created. Type: <code>ADMItemType</code> (see <code>ADMItem.h</code>)</td>
</tr>
<tr>
<td><code>inBoundsRect</code></td>
<td>Location and size of the item within the dialog's coordinate space. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inMilliseconds</td>
<td>Delay between events.</td>
</tr>
<tr>
<td>inAbortMask</td>
<td>Specifies actions that result in calling inAbortProc. Type: ADMActionMask (see ASTypes.h)</td>
</tr>
</tbody>
</table>
### Returns

An ADM timer.

### See also

`sADMDialog->AbortTimer()`

---

### sADMDialog->DefaultDraw()

Call ADM's default dialog draw function

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
</tbody>
</table>

### Returns

None.
See also

sADMDialog->GetDrawProc()
sADMDialog->SetDrawProc()

Example

```c
void doNothingDrawHandler(ADMDialogRef dialog, ADMDrawerRef drawer) {
    sADMDialog->DefaultDraw(dialog, drawer);
}
```

sADMDialog->DefaultNotify()

Call ADM's default dialog notification function

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inNotifier</td>
<td>An ADM notifier.</td>
</tr>
</tbody>
</table>

Returns

None.

Example

```c
void doNothingNotificationHandler(ADMDialogRef dialog, ADMNotifierRef notifier) {
    sADMDialog->DefaultNotify(dialog, notifier);
}
```

sADMDialog->DefaultTrack()

Call ADM's default dialog tracker function

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inTracker</td>
<td>An ADM tracker.</td>
</tr>
</tbody>
</table>

Returns

ture if tracker successfully called; false otherwise.

Example

```c
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMDialog->CreateItem()
sADMIItem->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

```c
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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inEnable</code></td>
<td>Pass <code>true</code> to enable the dialog and <code>false</code> to disable it.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMDialog->IsEnabled()`

---

sADMDialog->EndModal()  
Dismiss a modal dialog

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>
The ADM Dialog Suite

ADM Dialog Suite Functions

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 ASTypes.h) |

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

**Returns**

ID of the **Cancel** item in `inDialog`.

**See also**

`sADMDialog->SetCancelItemID()`

---

**sADMDialog->GetCursorID()**

Get the dialog's cursor ID

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>outPluginRef</code></td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td><code>outCursorID</code></td>
<td>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.)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMDialog->SetCursorID()`

---

**sADMDialog->GetDefaultItemID()**

Get the default item of a dialog

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

**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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

**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

```c
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 ASAPI (*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 ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

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.
The ADM Dialog Suite

ADM Dialog Suite Functions

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 ASAPI (*GetID)(ADMDialogRef inDialog);

Description

The GetID() function gets the ID of inDialog. This is the ID of the resource used to created 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.
The ADM Dialog Suite

ADM Dialog Suite Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inItemID</td>
<td>An ADM item ID. Can be a known item ID or it can be one of ADMs standard ADMItemID constants (see ADMDialog.h).</td>
</tr>
</tbody>
</table>

Returns

An ADM item in inDialog.

See also

sADMDialog->GetNextItem()

sADMDialog->GetLocalRect()

Get the size of a dialog

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>outLocalRect</td>
<td>The inDialog window in (0,0)-based coordinates. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMDialog->SetLocalRect()

sADMDialog->GetMask()

Get mask used for tracker function

```c
ADMActionMask ASAPI (*GetMask)(ADMDialogRef inDialog);
```
**Description**

The `GetMask()` function gets the mask used for controlling which events are received by the tracker.

**Parameters**

```plaintext
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

```c
ASInt32 ASAPI (*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**

```plaintext
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

```c
ASInt32 ASAPI (*GetMaxWidth)(ADMDialogRef inDialog);
```
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

**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

```c
ASInt32 ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

**Returns**

Minimum height (in pixels) of `inDialog`.

**See also**

- `sADMDialog->SetMinHeight()`
- `sADMDialog->GetMaxHeight()`
- `sADMDialog->SetMaxHeight()`
- `sADMDialog->GetMinWidth()`
- `sADMDialog->SetMinWidth()`
- `sADMDialog->GetMaxWidth()`
- `sADMDialog->SetMaxWidth()`
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.
Get notification data

```c
ADMUserData ASAPI (*GetNotifierData)(ADMDialogRef inDialog);
```

**Description**
The `GetNotifierData()` function gets the notification data of `inDialog` if any.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

**Returns**
The notifier data. Type: `ADMUserData` (see `ADMTypes.h`)

**See also**
- `sADMDialog->SetNotifierData()`

Get the ADM notification function being used for the dialog

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>
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

```c
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

```c
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.
The ADM Dialog Suite

ADM Dialog Suite Functions

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>outText</td>
<td>Buffer into which inDialog's text property is placed</td>
</tr>
<tr>
<td>inMaxLength</td>
<td>Size of outText.</td>
</tr>
</tbody>
</table>

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

```c
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

```c
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`:

```c
typedef struct GrafPort *ASWindowRef;
```

On Windows, this is the same as a Windows `HWND`:

```c
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

```c
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()

```c
void ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inInvalRect</code></td>
<td>A rectangle within <code>inDialog</code>. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMDialog->Invalidate()`

---

sADMDialog->IsActive()

```c
ASBoolean ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

**Returns**

`true` if `inDialog` is currently active; `false` otherwise.

**See also**

`sADMDialog->Activate()`
**sADMDialog->IsCollapsed()**

Determines whether a dialog palette is collapsed

```c
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

```c
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

```c
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()

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()

ASBoolean ASAPI (*IsUpdateEnabled)(ADMDialogRef inDialog);

Check whether the update enabled flag for this dialog is set.
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

```c
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

```c
void ASAPI (*LocalToScreenRect)(ADMDialogRef inDialog, ASRect* ioRect);
```
The LocalToScreenRect() function converts a rectangle in inDialog to its equivalent screen coordinates.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>ioRect</td>
<td>The rect. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPluginRef</td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td>inName</td>
<td>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.</td>
</tr>
<tr>
<td>inDialogID</td>
<td>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.</td>
</tr>
<tr>
<td>inDialogStyle</td>
<td>Style of dialog. See Dialog Basics: Styles. Type: ADMDialogStyle (see ADMDialog.h)</td>
</tr>
</tbody>
</table>
| 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->GetData() 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->GetData()
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inHorizPosition</code></td>
<td>New horizontal screen coordinate for <code>inDialog</code>.</td>
</tr>
<tr>
<td><code>inVertPosition</code></td>
<td>New vertical screen coordinate for <code>inDialog</code>.</td>
</tr>
</tbody>
</table>

Returns

None.

`sADMDialog->PopupMenu()` Pop up a modal dialog box

```c
ASInt32 ASAPI (*PopupModal)(SPPluginRef inPluginRef, const char* inName, ASInt32 inDialogID, ADMDialogInitProc inInitProc, ADMUserData inUserData, ASInt32 inOptions);
```

Description

The `PopupMenu()` function pops up a modal dialog box. The style is `kADMPopupControlDialogStyle`. There is no other choice for style.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inPluginRef</code></td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td><code>inName</code></td>
<td>Internal identifier for the dialog. Must be unique. The suggested convention is to use a leading prefix of your company's name, as in <code>MyCompanydialogxxxx</code>.</td>
</tr>
<tr>
<td><code>inDialogID</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
The ADM Dialog Suite
ADM Dialog Suite Functions

### Returns

ID of the item used to dismiss the dialog.

### See also

sADMDialog->Modal()

---

**sADMDialog->RegisterItemType()**

Add a custom item type

```c
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 `ADMCreatCustomItem` 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 ADMTypes.h)

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>ioPoint</td>
<td>The point. Type: ASPoint (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>ioRect</td>
<td>The rect. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.
See also

sADMDialog->LocalToScreenRect()

sADMDialog->SendNotify()  Send a notification to a dialog

```c
void ASAPI (*SendNotify)(ADMDialogRef inDialog, const char* inNotifierType);
```

**Description**

The `SendNotify()` function sends a notification of type `inNotifierType` to the `inDialog`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inNotifierType</td>
<td>The type of notification to send. (See <code>ADMNotifier.h</code> for the notification constants.)</td>
</tr>
</tbody>
</table>

**Returns**

None.

sADMDialog->SetBoundsRect()  Set the absolute position and size of a dialog

```c
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:

```c
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.

```c
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 \texttt{sADMDialog->Move()} function.

If the dialog is moved off-screen using this function, it will not be forced back onto the screen.

**Parameters**

\begin{itemize}
  \item \texttt{inDialog} An ADM dialog.
  \item \texttt{inBoundsRect} The size and position to make \texttt{inDialog} in screen coordinates. Type: \texttt{ASRect} (see \texttt{ASTypes.h})
\end{itemize}

**Returns**

None.

**See also**

- \texttt{sADMDialog->GetBoundsRect()}
- \texttt{sADMDialog->Move()}
- \texttt{sADMDialog->GetLocalRect()}
- \texttt{sADMDialog->SetLocalRect()}
- \texttt{sADMDialog->Size()}

\texttt{sADMDialog->SetCancelItemID()} Set the item to cancel a dialog

\begin{verbatim}
void ASAPI (*SetCancelItemID)(ADMDialogRef inDialog, ASInt32 inItemID);
\end{verbatim}

**Description**

The \texttt{SetCancelItemID()} function specifies which item in \texttt{inDialog} is the Cancel item. The Cancel item is notified when the \texttt{Esc} key is pressed. On the Macintosh platform, it is also notified when \texttt{Command} is pressed.

**Parameters**

\begin{itemize}
  \item \texttt{inDialog} An ADM dialog.
  \item \texttt{inItemID} ID of the Cancel item in \texttt{inDialog}.
\end{itemize}

**Returns**

None.

**See also**

- \texttt{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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inPluginRef</td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td>inCursorID</td>
<td>ID of the cursor resource. (See ADMResource.h for a list of built-in ADM cursor IDs.)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMDialog->GetCursorID()

sADMDialog->SetDefaultItemID() Set the default item of a dialog

void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inItemID</td>
<td>ID of the item in inDialog that is the default item.</td>
</tr>
</tbody>
</table>
Returns
None.

See also
sADMDialog->GetDefaultItemID()

---

**sADMDialog->SetDestroyProc()**

Set the ADM destroy function to use for the dialog

```
void ASAPI (*SetDestroyProc)(ADMDialogRef inDialog,
                           ADMDialogDestroyProc inDestroyProc);
```

**Description**
The `SetDestroyProc()` function assigns `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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inDestroyProc</code></td>
<td>A callback with the following signature (see ADMDialog.h):</td>
</tr>
<tr>
<td></td>
<td><code>ADMDialogDestroyProc (ADMDialogRef inDialog);</code></td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMDialog->GetDestroyProc()

---

**sADMDialog->SetDialogName()**

Set the dialog name

```
void ASAPI (*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.
The ADM Dialog Suite

ADM Dialog Suite Functions

Parameters

<table>
<thead>
<tr>
<th>inDialog</th>
<th>An ADM dialog.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inName</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMDialog->GetDialogName()
sADMDialogGroup->GetNamedDialog()

sADMDialog->SetDialogStyle()  Set the style of a dialog

void ASAPI (*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

<table>
<thead>
<tr>
<th>inDialog</th>
<th>An ADM Dialog.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialogStyle</td>
<td>The style of an ADM Dialog (see Dialog Basics: Styles) to which to set inDialog.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inDrawProc</code></td>
<td>A callback with the following signature (see <code>ADMDialog.h</code>):</td>
</tr>
</tbody>
</table>
|               | ```c
|               | 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

```c
void ASAPI (*SetFont)(ADMDialogRef inDialog, ADMFont inFont);
```

Description

The `SetFont()` function sets a dialog’s text font to the indicated font style.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inFont</code></td>
<td>Font to use. Type: <code>ADMFont</code> (see <code>ADMTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMDialog->GetFont()`
**sADMDialog->SetForcedOnScreen()**  
Force the dialog on screen

```c
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

```c
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

```c
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**

<table>
<thead>
<tr>
<th>inDialog</th>
<th>An ADM dialog.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inLocalRect</td>
<td>Size of the local rect. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**
None.

**See also**
- `sADMDialog->GetLocalRect()`
- `sADMDialog->Size()`
- `sADMDialog->SetBoundsRect()`
- `sADMDialog->GetBoundsRect()`

---

**sADMDialog->SetMask()**

Set mask for use with tracker function

```c
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inWidth</td>
<td>Maximum width (in pixels) of resizable dialog inDialog.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inHeight</td>
<td>Minimum height (in pixels) of resizable dialog inDialog.</td>
</tr>
</tbody>
</table>
sADMDialog->SetMinWidth()  Get the minimum width of a dialog 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>inDialog</em></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><em>inWidth</em></td>
<td>Minimum width (in pixels) of resizable dialog <em>inDialog</em>.</td>
</tr>
</tbody>
</table>

Returns
None.

See also
- sADMDialog->GetMinHeight()
- sADMDialog->GetMaxHeight()
- sADMDialog->SetMaxHeight()
- sADMDialog->GetMinWidth()
- sADMDialog->SetMinWidth()
- sADMDialog->GetMaxWidth()
- sADMDialog->SetMaxHeight()
sADMDialog->SetNotifierData() Set notification data

```c
void ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inUserData</code></td>
<td>Custom notification data. Type: <code>ADMUserData</code> (see <code>ADMTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMDialog->GetNotifierData()`

---

sADMDialog->SetNotifyProc() Set the ADM notification function to use for the dialog

```c
void ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>
| `inNotifyProc` | A callback with the following signature (see `ADMDialog.h`):
|                | `ADMDialogNotifyProc (ADMDialogRef inDialog, 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
sADMDialog->GetNotifyProc()

sADMDialog->SetText()
Set the dialog title

```c
void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inText</code></td>
<td>Text for <code>inDialog</code>'s title.</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMDialog->GetText()
sADMDialog->GetTextLength()

sADMDialog->SetTrackProc()
Set the ADM tracker function to use for the dialog

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>
## inTrackProc

A callback with the following signature (see `ADMDialog.h`):

```c
ASBoolean ADMDialogTrackProc (ADMDialogRef inDialog, 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 except for keystroke events. In those cases it means whether or not the keystroke was handled.

### Returns

None.

### See also

`sADMDialog->GetTrackProc()`

## sADMDialog->SetUpdateEnabled()

Set the update enabled flag for this dialog

```c
void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inEnableUpdate</code></td>
<td>If set to <code>true</code>, update-enabled flag is set for <code>inDialog</code>. If set to <code>false</code>, it is disabled.</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

`sADMDialog->IsUpdateEnabled()`
**sADMDiallog->SetUserData()**

Set the user data pointer for a dialog

```c
void ASAPI (*SetUserData)(ADMDialogRef inDialog, ADMUserData inUserData);
```

**Description**

The `SetUserData()` function sets the 4-byte user value stored with `inDialog`. It is automatically set by the `sADMDiallog->Create()` and `sADMDiallog->Modal()` functions.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>ADMUserData</code></td>
<td>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 <code>sADMDiallog-&gt;GetUserData()</code> function. Type: <code>ADMUserData</code> (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMDiallog->GetUserData()`, `sADMDiallog->Create()`, `sADMDiallog->Modal()`

---

**sADMDiallog->SetVerticalIncrement()**

Set the vertical increment to use when resizing a window

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inIncrement</td>
<td>The amount that will be added to or subtracted from the dialog's height as the user increases or decreases its size.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- sADMDialog->GetVerticalIncrement()
- sADMDialog->GetHorizontalIncrement()
- sADMDialog->SetHorizontalIncrement()

sADMDialog->SetWindowRef()

Set the window reference

```c
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`:

```c
typedef struct GrafPort *ASWindowRef;
```

On Windows, this is the same as a Windows `HWND`:

```c
typedef void * ASWindowRef;
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inWindowRef</td>
<td>Platform window reference for <code>inDialog</code>. Type: <code>ASWindowRef</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- sADMDialog->GetWindowRef()
sADMDialog->Show()

Hide or show a dialog

```c
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

```c
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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

**Returns**

The help ID. Type: **ASHelpID** (See ASHelp.h)

**See also**

- `sADMDialog->SetHelpID()`

---

### sADMDialog->Help()

Calls the help routine for a dialog

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

**Returns**

None.
sADMDiallog->SetHelpID()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>inHelpID</td>
<td>The resource ID for the ASHelp resource. Type: ASHelpID (See ASHelp.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMDiallog->GetHelpID()
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outCount</td>
<td>The number of dialogs that ADM currently knows about.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM docked/tabbed dialog.</td>
</tr>
<tr>
<td>outGroupName</td>
<td>inDialog's group name. See The ADM Dialog Group Suite's Position Code and Group Name.</td>
</tr>
</tbody>
</table>
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()

ASErr 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDialog</td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td>outName</td>
<td>Reference to the unique character ID name of inDialog.</td>
</tr>
</tbody>
</table>

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()

ASErr ASAPI (*GetNamedDialog) (const char* inName, ADMDialogRef* outDialog);

Description

The GetNamedDialog() function obtains outDialog specified by its unique dialog name.

Adobe Dialog Manager Programmer’s Guide and Reference
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inName</td>
<td>Reference to the unique character ID name of <code>inDialog</code>.</td>
</tr>
<tr>
<td>outDialog</td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inIndex</td>
<td>0-based index used for iterating through ADM's current list of dialogs.</td>
</tr>
<tr>
<td>outDialog</td>
<td>Reference to an ADM dialog in ADM's current list of dialogs.</td>
</tr>
</tbody>
</table>

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

```c
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

Position Code 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

```c
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

Position Code 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

```c
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

```cpp
sADMDialogGroup->GetDialogGroupInfo(myDialog, &buffer, &positionCode);
isFrontTab = sADMDialogGroup->IsFrontTab(positionCode);
```

`sADMDialogGroup->IsStandalonePalette()`

Find out whether palette is docked or not

```cpp
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

```cpp
sADMDialogGroup->GetDialogGroupInfo(myDialog, &buffer, &positionCode);
isStandalone = sADMDialogGroup->IsStandalonePalette(positionCode);
```

`sADMDialogGroup->SetDialogGroupInfo()`

Set group name and position code

```cpp
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM docked/tabbed dialog.</td>
</tr>
<tr>
<td><code>inGroupName</code></td>
<td><code>inDialog</code>’s group name. See The ADM Dialog Group Suite’s Position Code and Group Name.</td>
</tr>
</tbody>
</table>

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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inTabGroupName</code></td>
<td>The tab group for <code>inDialog</code> to join.</td>
</tr>
<tr>
<td><code>inBringToFront</code></td>
<td>If <code>true</code>, <code>inDialog</code> is brought to front; if <code>false</code>, <code>inDialog</code> is not brought to front.</td>
</tr>
</tbody>
</table>

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()

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()

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()

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()`
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:

```c
#define kADMDrawerSuite "ADM Drawer Suite"
```

with the version constant:

```c
#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:

```c
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:

```c
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:

```c
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 the each of the ADM drawer functions:

```c
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 the ADMColor 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`):

```c
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.

![Diagram of drawing modes]

**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 affect.

- Horizontal lines
- Horizontal lines with 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

```c
void ASAPI (*Clear)(ADMDrawerRef inDrawer);
```

**Description**

The `clear()` function fills the area of the ADM object being drawn with the background color.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**sADMDrawer->ClearRect()**

Clear a rect to the background color

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>The rectangle whose area is to be filled with the background color. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.
The ADM Drawer Suite

ADM Drawer Suite Functions

### sADMDrawer->Create()

Create an ADM drawer reference

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inPortRef</code></td>
<td>The port reference. Obtained using <code>sADMDrawer-&gt;GetADMWindowPort()</code>. Type: ASPortRef (see ASTypes.h)</td>
</tr>
<tr>
<td><code>inBoundsRect</code></td>
<td>The rect into which to draw. Type: ASRect (see ASTypes.h)</td>
</tr>
<tr>
<td><code>inFont</code></td>
<td>The font to use. Type: ADMFont (see ADMTypes.h)</td>
</tr>
<tr>
<td><code>inForceRoman</code></td>
<td>Used to force Roman font on non-roman OS.</td>
</tr>
</tbody>
</table>

**Returns**

An ADM drawer.

**See also**

- `sADMDrawer->Destroy()`
- `sADMDialog->GetWindowRef()`
- `sADMDrawer->GetADMWindowPort()`
- `sADMDrawer->GetPortRef()`
- `sADMDrawer->GetUpdateRect()`

### sADMDrawer->Destroy()

Destroy an ADM drawer

```c
void ASAPI (*Destroy)(ADMDrawerRef inDrawer);
```

**Description**

The `Destroy()` function destroys an `ADMDrawerRef` created with `sADMDrawer->Create()`.

---

Adobe Dialog Manager Programmer’s Guide and Reference
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>The ADM drawer to be destroyed.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMDrawer->Create()

sADMDrawer->DrawADMImage()

draw an ADM image

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inImage</td>
<td>An image created using functions in the ADM Image suite—sADMImage-&gt;Create(), sADMImage-&gt;CreateBitmap(), or sADMImage-&gt;CreateOffscreen().</td>
</tr>
<tr>
<td>inTopLeftPoint</td>
<td>Anchor point for <code>inImage</code>. Type: <code>ASPoint</code> (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

sADMDrawer->DrawADMImageCentered()

draw a centered ADM image

```c
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 `ASTypes.h`)

Returns

- None.

### `sADMDrawer->DrawAGMImage()`

#### Draw an AGM Image

```c
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 `ASTypes.h`)

- **inFlags**
  Currently not implemented. Always **0**.

Returns

- None.

See also

- `sADMDrawer->GetAGMPort()`
sADMDrawer->DrawDownArrow()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>Rect into which to draw the arrow. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMDrawer->DrawLeftArrow()`
- `sADMDrawer->DrawRightArrow()`
- `sADMDrawer->DrawUpArrow()`
- `sADMDrawer->DrawIcon()`

sADMDrawer->DrawIcon()

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inIcon</code></td>
<td>The icon to draw. Usually loaded using <code>sADMItem-&gt;GetFromResource()</code></td>
</tr>
<tr>
<td><code>inTopLeftPoint</code></td>
<td>The anchor point for the top-left corner of the icon</td>
</tr>
</tbody>
</table>

**Returns**

None.
The ADM Drawer Suite

ADM Drawer Suite Functions

Returns

None.

See also

sADMDrawer->DrawRecoloredIcon()
sADMDrawer->DrawRecoloredIconCentered()
sADMDrawer->DrawIconCentered()
sADMItem->GetPluginRef()
sADMIcon->GetFromResource()

Example

```c
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inIcon</td>
<td>The icon to draw. Loaded using <code>sADMIcon-&gt;GetFromResource()</code>. See <code>sADMDrawer-&gt;DrawIcon()</code>.</td>
</tr>
<tr>
<td>inRect</td>
<td>The rect to whose center <code>inIcon</code> is anchored. Type: <code>ASRect</code> (see ASTypes.h)</td>
</tr>
</tbody>
</table>
Returns

None.

See also

- sADMDrawer->DrawRecoloredIcon()
- sADMDrawer->DrawRecoloredIconCentered()
- sADMDrawer->DrawIcon()
- sADMItem->GetPluginRef()
- sADMIcon->GetFromResource()

sADMDrawer->DrawLeftArrow()  

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()  

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inStartPoint</td>
<td>Starting point of line to be drawn. Type: ASPoint (see ASTypes.h)</td>
</tr>
<tr>
<td>inEndPoint</td>
<td>Ending point of line to be drawn. Type: ASPoint (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

**sADMDrawer->DrawOval()**

Draw an unfilled oval

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inRect</td>
<td>Bounding rectangle for the oval to be drawn. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- `sADMDrawer->FillOval()`

**sADMDrawer->DrawPolygon()**

Draw an unfilled polygon

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inPoints</code></td>
<td>An array of pointers to the points of the polygon to be drawn. Type: <code>ASPoint</code> (see <code>ASTypes.h</code>)</td>
</tr>
<tr>
<td><code>inNumPoints</code></td>
<td>Zero-based index indicating the number of elements in <code>inPoints</code>.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMDrawer->SubtractClipPolygon()`
- `sADMDrawer->FillPolygon()`
- `sADMDrawer->IntersectClipPolygon()`
- `sADMDrawer->SetClipPolygon()`
- `sADMDrawer->UnionClipPolygon()`

---

**`sADMDrawer->DrawRaisedRect()`**

```c
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.

![A Raised Rectangle](image)

**Figure 9.3**  
A Raised Rectangle

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
</tbody>
</table>
The ADM Drawer Suite

ADM Drawer Suite Functions

**sADMDrawer->DrawRecoloredIcon()**

```c
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 ASTypes.h)
- **inStyle**
  Style for the recolored icon. Type: `ADMRecolorStyle` (see ADMDrawer.h)

**Returns**

None.

**See also**

- `sADMDrawer->DrawRecoloredIconCentered()`
- `sADMDrawer->DrawIcon()`
- `sADMDrawer->DrawIconCentered()`
- `sADMItem->GetPluginRef()`
- `sADMIcon->GetFromResource()`
**sADMDrawer->DrawRecoloredIconCentered()**

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inIcon</code></td>
<td>The icon to draw. Usually loaded using <code>sADMDrawer-&gt;GetPluginRef()</code>. See <code>sADMDrawer-&gt;DrawIcon()</code>.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>The rect to whose center <code>inIcon</code> is anchored. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
<tr>
<td><code>inStyle</code></td>
<td>Style for the recolored icon. Type: <code>ADMRecolorStyle</code> (see <code>ADMDrawer.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMDrawer->DrawRecoloredIcon()`
- `sADMDrawer->DrawIcon()`
- `sADMDrawer->DrawIconCentered()`
- `sADMItem->GetPluginRef()`
- `sADMIcon->GetFromResource()`
- `sADMDrawer->DrawRecoloredResPicture()`

**sADMDrawer->DrawRecoloredResPicture()**

```c
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

```c
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()`. 
The ADM Drawer Suite

ADM Drawer Suite Functions

**DrawRect**

```
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>The rectangle to be drawn. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMDrawer->FillRect()`
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 `sADMDrawer->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

```c
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**

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inPluginRef</code></td>
<td>Plug-in reference. Usually obtained using <code>sADMItem-&gt;GetPluginRef()</code> or <code>sADMDialog-&gt;GetPluginRef()</code>.</td>
</tr>
<tr>
<td><code>inRsrcID</code></td>
<td>Resource ID. On Macintosh, the resource type is ‘PICT’ and on Windows it is a bitmap resource.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>The rect to whose center the picture is anchored. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMDrawer->DrawRecoloredIcon()`  
- `sADMDrawer->DrawRecoloredIconCentered()`  
- `sADMDrawer->DrawResPicture()`  
- `sADMDrawer->DrawRecoloredResPicture()`  
- `sADMDrawer->DrawRecoloredResPictureCentered()`  
- `sADMDrawer->GetResPictureBounds()`

---

sADMDrawer->DrawRightArrow()

**Draw a triangular right arrow**

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>Rect into which to draw the arrow. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

- `sADMDrawer->DrawLeftArrow()`
- `sADMDrawer->DrawDownArrow()`
- `sADMDrawer->DrawUpArrow()`
- `sADMDrawer->DrawSunkenRect()`

### `sADMDrawer->DrawSunkenRect()`

**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.

```c
void ASAPI (*DrawSunkenRect)(ADMDrawerRef inDrawer, const ASRect* inRect);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>The size and placement of the sunken rectangle. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>
Returns
None.

See also
sADMDrawer->DrawRaisedRect()

sADMDrawer->DrawText()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inText</td>
<td>C-style string to be drawn.</td>
</tr>
<tr>
<td>inPoint</td>
<td>The point at which to start drawing the text.</td>
</tr>
<tr>
<td></td>
<td>Specified in inDrawer's coordinate space.</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMDrawer->DrawTextCentered()
sADMDrawer->DrawTextInaBox()
sADMDrawer->DrawTextLeft()
sADMDrawer->DrawTextRight()

sADMDrawer->DrawTextCentered()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inText</td>
<td>C-style string to be drawn. If the length of the text is greater than the width of inRect, the text will be clipped.</td>
</tr>
<tr>
<td>inRect</td>
<td>The rect into which inText is drawn, center-justified. Specified in inDrawer's coordinate space. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMDrawer->DrawText()
sADMDrawer->DrawTextInaBox()
sADMDrawer->DrawTextLeft()
sADMDrawer->DrawTextRight()

sADMDrawer->DrawTextInaBox()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inRect</td>
<td>The box into which to flow inText. Type: ASRect (see ASTypes.h)</td>
</tr>
<tr>
<td>inText</td>
<td>The text to flow into inRect.</td>
</tr>
</tbody>
</table>

Returns

None.
See also

sADMDrawer->DrawText()
sADMDrawer->DrawTextCentered()
sADMDrawer->DrawTextLeft()
sADMDrawer->DrawTextRight()

sADMDrawer->DrawTextLeft()
Draw left justified text in a rectangle

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inText</code></td>
<td>C-style string to be drawn. If the length of the text is greater than the width of <code>inRect</code>, the text will be clipped.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>The rect into which <code>inText</code> is drawn, left-justified. Specified in <code>inDrawer</code>'s coordinate space. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

See also

sADMDrawer->DrawText()
sADMDrawer->DrawTextCentered()
sADMDrawer->DrawTextLeft()
sADMDrawer->DrawTextRight()

sADMDrawer->DrawTextRight()
Draw right justified text in a rectangle

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inText</code></td>
<td>C-style string to be drawn. If the length of the text is greater than the width of <code>inRect</code>, the text will be clipped.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>The rect into which <code>inText</code> is drawn, right-justified. Specified in <code>inDrawer</code>'s coordinate space. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- `sADMDrawer->DrawText()`
- `sADMDrawer->DrawTextCentered()`
- `sADMDrawer->DrawTextInaBox()`
- `sADMDrawer->DrawTextLeft()`

`sADMDrawer->DrawUpArrow()`

Draw a triangular up arrow

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>Rect into which to draw the arrow. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>
sADMDrawer->FillOval()

Draw a filled oval

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>Bounding rectangle for the oval to be filled. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- sADMDrawer->DrawLeftArrow()
- sADMDrawer->DrawDownArrow()
- sADMDrawer->DrawRightArrow()

---

sADMDrawer->FillPolygon()

Draw a filled polygon

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
</tbody>
</table>
### sADMDrawer->FillRect()

**Description**

The `FillRect()` function draws a filled-in rectangle in the current color for `inRect`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>The rectangle to be drawn. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMDrawer->DrawRect()`

---

### sADMDrawer->GetADMColor()

**Description**

The `GetADMColor()` function returns the constant of the ADM color currently being used for drawing.

```c
ADMColor ASAPI (*GetADMColor)(ADMDrawerRef inDrawer);
```
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>outBoundsRect</code></td>
<td>Bounds of the drawer's object relative to the origin. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.
Example

```c
//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

```c
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**

<table>
<thead>
<tr>
<th>inDrawer</th>
<th>An ADM drawer.</th>
</tr>
</thead>
<tbody>
<tr>
<td>outClipRect</td>
<td>The clipping rectangle. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMDrawer->GetClipRect()`
- `sADMDrawer->SetClipRect()`
- `sADMDrawer->SetClipPolygon()`
- `sADMDrawer->SubtractClipRect()`
- `sADMDrawer->UnionClipRect()`

`sADMDrawer->GetDrawMode()` Get the current drawing mode

```c
ADMDrawMode ASAPI (*GetDrawMode)(ADMDrawerRef inDrawer);
```

**Description**

The `GetDrawMode()` function returns the current drawing mode of the drawer.

**Parameters**

<table>
<thead>
<tr>
<th>inDrawer</th>
<th>An ADM drawer.</th>
</tr>
</thead>
</table>
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 ADMDrawer.h) |

**Returns**

None.
sADMDrawer->GetOrigin()

Get the drawer origin

void ASAPI (*GetOrigin)(ADMDrawerRef inDrawer, APoint* 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>outOrigin</td>
<td>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: <code>APoint</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
</tbody>
</table>

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 ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inPluginRef</td>
<td>Plug-in reference. Usually obtained using sADMItem-&gt;GetPluginRef() or sADMDialog-&gt;GetPluginRef().</td>
</tr>
<tr>
<td>inRsrcID</td>
<td>Resource ID. On Macintosh, the resource type is ‘PICT’ and on Windows it is a bitmap resource.</td>
</tr>
<tr>
<td>outBoundsRect</td>
<td>Bounds of inRsrcID. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMDrawer->DrawRecoloredResPicture()
sADMDrawer->DrawRecoloredResPictureCentered()
sADMDrawer->DrawResPicture()
sADMDrawer->DrawResPictureCentered()

sADMDrawer->GetRGBColor()  Get the values of the RGB color being used

void ASAPI (*GetRGBOColor)(ADMDrawerRef inDrawer, ASRGBColor* outColor);

Description

The GetRGBOColor() 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>outColor</code></td>
<td>The RGB values of the color currently being used for drawing. Type: <code>ASRGBColor</code> (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- `sADMDrawer->SetRGBColor()`
- `sADMDrawer->GetADMColor()`
- `sADMDrawer->SetADMColor()`

### `sADMDrawer->GetTextRectHeight()`

Get the height of the text rect

```c
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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inText</code></td>
<td>The text string to be measured.</td>
</tr>
</tbody>
</table>

**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

```c
void ASAPI (*GetThisFontInfo)(ADMFont inFont, ADMFontInfo* outFontInfo);
```

**Description**

The `GetThisFontInfo()` function obtains metrics for the specified font.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inFont</code></td>
<td>The font for which metrics are obtained. Type: <code>ADMFont</code> (see <code>ADMTypes.h</code>)</td>
</tr>
<tr>
<td><code>outFontInfo</code></td>
<td>The font metrics for the current font. Type: <code>ADMFontInfo</code> (see <code>ADMDrawer.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

---

**sADMDrawer->GetUpdateRect()**

Return rect that needs to be updated

```c
void ASAPI (*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).
**sADMDrawer->IntersectClipPolygon()**

Intersect the current clipping area with a polygon

```c
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->Create()`
- `sADMDrawer->SubtractClipPolygon()`
- `sADMDrawer->DrawPolygon()`
- `sADMDrawer->FillPolygon()`
- `sADMDrawer->SetClipPolygon()`
- `sADMDrawer->UnionClipPolygon()`
sADMDrawer->IntersectClipRect()

Intersect the current clipping area with a new rectangle

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inClipRect</code></td>
<td>Rectangle used to create the new clipping area. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

sADMDrawer->InvertRect()

Invert the colors of a rectangle

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDrawer</code></td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>The rect whose colors are to be inverted. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.
sADMDrawer->ReleaseADMWindowPort()

Release an ADM window’s port reference

```c
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

```c
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

```c
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

```c
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 ASTypes.h)

**Returns**
None.

**See also**
- sADMDrawer->GetClipRect()
- sADMDrawer->SubtractClipRect()
- sADMDrawer->UnionClipRect()

---

#### sADMDrawer->SetDrawMode()
Set the drawing mode

```c
void ASAPI (*SetDrawMode)(ADMDrawerRef inDrawer, ADMDrawMode inDrawMode);
```

**Description**
The `SetDrawMode()` function sets the drawing mode to be used by future drawing commands.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inDrawMode</td>
<td>Specifies the drawing mode of the drawer—normal or XOR (see Drawing Modes). Type: ADMDrawMode (see ADMDrawer.h).</td>
</tr>
</tbody>
</table>

**Returns**
None.

**See also**
- sADMDrawer->GetDrawMode()

---

#### sADMDrawer->SetFont()
Set the font to be used

```c
void ASAPI (*SetFont)(ADMDrawerRef inDrawer, ADMFont inFont);
```

**Description**
The `SetFont()` function sets the font to be used for subsequent text operations.
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inFont</td>
<td>Sets the font for use. Type: <code>ADMFont</code> (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMDrawer->GetFont()`

---

**sADMDrawer->SetOrigin()**

Set the drawer origin

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inOrigin</td>
<td>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: <code>ASPoint</code> (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMDrawer->GetOrigin()`

---

**sADMDrawer->SetRGBColor()**

Set the RGB color to use

```c
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()

**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->SubtractClipRect()

Remove a clip rectangle from the existing clip area

```c
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
<tr>
<td>inClipRect</td>
<td>Rectangle to be removed from the current clipping area. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMDrawer->GetClipRect()`
- `sADMDrawer->SetClipRect()`
- `sADMDrawer->UnionClipRect()`

---

sADMDrawer->UnionClipPolygon()

Combine the existing clip area with a polygon

```c
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
</tbody>
</table>
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 ASTypes.h)

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

```c
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 ASTypes.h)

**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 kADMEntrySuite "ADM Entry Suite"
```

with the version constant:

```
#define kADMEntrySuiteVersion2 2
```

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

The suite is acquired as follows:

```
ADMEntrySuite *sADMEntry;
error = sSPBasic->AcquireSuite(kADMEntrySuite, kADMEntrySuiteVersion2, &sADMEntry);
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 `ADMEntryInitProc` (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

sADMEntry->AbortTimer()

Abort timer

void ASAPI (*AbortTimer)(ADMEntryRef inEntry, ADMTimerRef inTimer);

Description

The AbortTimer() function aborts a timer procedure. It is used if the event specified by the inAbortMask in sADMEntry->CreateTimer() occurs or if you destroy your dialog before your timer expires.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>inTimer</td>
<td>An ADM timer.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMEntry->CreateTimer()

sADMEntry->Activate()

Make an entry active or inactive

void ASAPI (*Activate)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>
The ADM Entry Suite
ADM Entry Suite Functions

ADM Entry Suite Functions

Returns

None.

See also

sADMEntry->IsActive()

sADMEntry->Check()

Set whether an entry is checked

void ASAPI (*Check)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>inCheck</td>
<td>If true, the entry has a checkmark; if false, it does not have a checkmark.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMEntry->IsChecked()

sADMEntry->Create()

Create a new ADM entry

ADMEntryRef 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 sADMEntry->SetID() function. Use sADMEntry->Destroy() to dispose of the entry.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM list.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>inMilliseconds</code></td>
<td>Number of milliseconds in the delay.</td>
</tr>
<tr>
<td><code>inAbortMask</code></td>
<td>Actions that cause a timer abort. If an action occurs, <code>inTimerAbortProc</code> is called. Maskable actions are listed in <code>ADMTracker.h</code>. Type: <code>ADMActionMask</code> (see <code>ADMTypes.h</code>)</td>
</tr>
<tr>
<td><code>inTimerProc</code></td>
<td>A callback with the following signature (see <code>ADMEntry.h</code>): <code>ASBoolean ADMEntryTimerProc(ADMEntryRef inEntry, ADMTimerRef inTimer);</code>. Executed if the delay succeeds. Returns a boolean. If it returns <code>true</code>, then <code>inTimerProc</code> will be called again after <code>inMilliseconds</code>. If it returns <code>false</code> then <code>inTimerProc</code> will no longer be called.</td>
</tr>
<tr>
<td><code>inTimerAbortProc</code></td>
<td>A callback with the following signature (see <code>ADMEntry.h</code>): <code>ADMEntryTimerAbortProc(ADMEntryRef inEntry, ADMTimerRef inTimer, ADMAction inAbortAction);</code>. Called if an action specified by <code>inAbortMask</code> occurs. The values for <code>inAbortAction</code> are of type <code>ADMAction</code> and are listed in <code>ADMTracker.h</code>.</td>
</tr>
<tr>
<td><code>inOptions</code></td>
<td>Currently unused. Always pass 0 (zero).</td>
</tr>
</tbody>
</table>
**Returns**

The new `ADMTimerRef`.

**See also**

`sADMEntry->AbortTimer()`

---

**sADMEntry->DefaultDraw()**

Call ADM's default entry draw function

```c
void ASAPI (*DefaultDraw)(ADMEntryRef 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**

```c
void doNothingDrawHandler(ADMEntryRef inEntry, ADMDrawerRef inDrawer) {
    sADMEntry->DefaultDraw(entry, drawer);
}
```

---

**sADMEntry->DefaultNotify()**

Call an ADM entry’s default notification function

```c
void ASAPI (*DefaultNotify)(ADMEntryRef inEntry, ADMNotifierRef inNotifier);
```

**Parameters**

- `inEntry` An ADM entry.
- `inNotifier` An ADM notifier.

**Returns**

None.

**See also**

`sADMList->SetNotifyProc()`

**Example**

```c
void doNothingNotifyHandler(ADMEntryRef inEntry, ADMNotifierRef inNotifier) {
    sADMEntry->DefaultNotify(entry, notifier);
}
```
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>inNotifier</code></td>
<td>An ADM notifier.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMList->SetNotifyProc()`

Example

```c
void doNothingNotificationHandler(ADMEntryRef entry, ADMNotifierRef notifier) {
    sADMEntry->DefaultNotify(entry, notifier);
    // Custom behavior would go here...
}
```

**sADMEntry->DefaultTrack()**  
Call the default tracker function for the ADM entry

```c
ASBoolean ASAPI (*DefaultTrack)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>
The ADM Entry Suite

ADM Entry Suite Functions

inTracker

An ADM tracker.

Returns

true if tracker successfully called; false otherwise.

See also

sADMList->SetTrackProc()

Example

ASBoolean doNothingTrackHandler(ADMEntryRef entry, ADMTrackerRef tracker) {
    return sADMEntry->DefaultTrack(entry, tracker);
}

sADMEntry->Destroy()

Remove an ADM entry from a list

void ASAPI (*Destroy)(ADMEntryRef 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()
sADMEntry->Create()

sADMEntry->Enable()

Enable or disable an entry

void ASAPI (*Enable)(ADMEntryRef inEntry, ASBoolean inEnable);
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>inEnable</td>
<td>If true, the entry is enabled; if false, it is disabled.</td>
</tr>
</tbody>
</table>

**Returns**

None.

See also

sADMEntry->IsEnabled()

---

**sADMEntry->GetBoundsRect()**

Get the absolute position and size of an entry

```c
void ASAPI (*GetBoundsRect)(ADMEntryRef inEntry, ASRect* outBoundsRect);
```

**Description**

The GetBoundsRect() function gets the current size and position of inEntry in its containing list's coordinate space.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>outBoundsRect</td>
<td>Current size and position of inEntry in its containing list's coordinate space. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

---

**sADMEntry->GetCheckGlyph()**

Get check glyph

```c
ADMStandardCheckGlyphID ASAPI (*GetCheckGlyph)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

Returns

The type of glyph set for \texttt{inEntry}. Type: \texttt{ADMStandardCheckGlyphID} (see \texttt{ADMEntry.h}).

See also

\texttt{sADMEntry->SetCheckGlyph()}

\texttt{sADMEntry->GetDisabledPicture()}

Get the picture to display when entry is disabled

\texttt{ADMIconRef ASAPI (*GetDisabledPicture)(ADMEntryRef inEntry);}  

Description

The \texttt{GetDisabledPicture()} function returns the picture that is set to display when \texttt{inEntry} is disabled.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

Returns

An ADM icon.

See also

\texttt{sADMEntry->SetDisabledPicture()}
\texttt{sADMEntry->GetSelectedPicture()}
\texttt{sADMEntry->GetSelectedPictureID()}
\texttt{sADMEntry->GetPicture()}
\texttt{sADMEntry->GetPictureID()}
\texttt{sADMEntry->GetDisabledPictureID()}

\texttt{sADMEntry->GetDisabledPictureID()}

Get the entry's disabled picture ID

\texttt{void ASAPI (*GetDisabledPictureID)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>outPictureResID</code></td>
<td>Resource ID for the picture. If the entry does not have a disabled picture, returns 0.</td>
</tr>
<tr>
<td><code>outPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- `sADMEntry->SetDisabledPictureID()`
- `sADMEntry->GetSelectedPicture()`
- `sADMEntry->GetSelectedPictureID()`
- `sADMEntry->GetPicture()`
- `sADMEntry->GetPictureID()`
- `sADMEntry->GetDisabledPicture()`
- `sADMEntry->GetID()`

---

**sADMEntry->GetID()**

Get the ID of an entry

```c
ASInt32 ASAPI (*GetID)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

Returns

ID of `inEntry`.

See also

- `sADMEntry->SetID()`
**sADMEntry->GetIndex()**

Get the index of an entry

```c
ASInt32 ASAPI (*GetIndex)(ADMEntryRef 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**

```c
ASInt32 GetListValue(ADMItem theListItem) {
    ADMList theList = sADMItem->GetList(theListItem);
    ADMEntry theEntry = sADMList->GetActiveEntry(theItemList);
    return sADMEntry->GetIndex(theEntry);
}
```

**sADMEntry->GetList()**

Get the list of an entry

```c
ADMListRef ASAPI (*GetList)(ADMEntryRef 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.

**sADMEntry->GetLocalRect()**

Get the size of an entry

```c
void ASAPI (*GetLocalRect)(ADMEntryRef inEntry, ASRect* outLocalRect);
```
Description

The `GetLocalRect()` function gets the size of `inEntry` in (0,0)-based dimensions.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>outLocalRect</code></td>
<td>The size of <code>inEntry</code> in (0,0)-based dimensions. The <strong>bottom</strong> and <strong>right</strong> members of the <code>ASRect</code> structure are the entry’s size. Type: <code>ASRect</code> (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

`sADMEntry->GetPicture()` — Get the picture

void ASAPI (*GetPicture)(ADMEntryRef inEntry);

Description

The `GetPicture()` function returns the picture used to draw `inEntry`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

Returns

An ADM icon.

See also

- `sADMEntry->SetPicture()`
- `sADMEntry->GetPictureID()`
- `sADMEntry->GetSelectedPicture()`
- `sADMEntry->GetSelectedPictureID()`
- `sADMEntry->GetDisabledPicture()`
- `sADMEntry->GetDisabledPictureID()`

`sADMEntry->GetPictureID()` — Get the entry’s picture ID

void ASAPI (*GetPictureID)(ADMEntryRef inEntry, ASInt32* outPictureResID, const char** outPictureResName);
The ADM Entry Suite

ADM Entry Suite Functions

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>outPictureResID</code></td>
<td>Resource ID for the picture. If the entry does not have a picture, returns 0.</td>
</tr>
<tr>
<td><code>outPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- `sADMEntry->SetPictureID()`
- `sADMEntry->GetPicture()`
- `sADMEntry->GetSelectedPicture()`  
- `sADMEntry->GetSelectedPictureID()`  
- `sADMEntry->GetDisabledPicture()`  
- `sADMEntry->GetDisabledPictureID()`

`sADMEntry->GetSelectedPicture()`  

Get the selected picture

```c
ADMIIconRef ASAPI (*GetSelectedPicture)(ADMEntryRef inEntry);
```

Description

The `GetSelectedPicture()` function gets the picture to be displayed when `inEntry` is selected.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

Returns

An ADM icon.

See also

- `sADMEntry->SetSelectedPicture()`  
- `sADMEntry->GetSelectedPictureID()`  
- `sADMEntry->GetPicture()`  
- `sADMEntry->GetPictureID()`
**sADMEntry->GetSelectedPictureID()**

Get the entry’s selected picture ID

```c
void ASAPI (*GetSelectedPictureID)(ADMEntryRef 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>outPictureResID</code></td>
<td>Resource ID for the picture. If the entry does not have a disabled picture, returns 0.</td>
</tr>
<tr>
<td><code>outPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMEntry->SetSelectedPictureID()`
- `sADMEntry->GetSelectedPicture()`
- `sADMEntry->GetPicture()`
- `sADMEntry->GetPictureID()`
- `sADMEntry->GetDisabledPicture()`
- `sADMEntry->GetDisabledPictureID()`

**sADMEntry->GetText()**

Get the entry’s text

```c
void ASAPI (*GetText)(ADMEntryRef 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`. 
The ADM Entry Suite

ADM Entry Suite Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>outText</td>
<td>Buffer for inEntry’s text.</td>
</tr>
<tr>
<td>inMaxLength</td>
<td>Size of the buffer.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMEntry->SetText()

sADMEntry->GetTextLength()

Get the length of the entry’s text

ASInt32 ASAPI (*GetTextLength)(ADMEntryRef inEntry);

Description

The GetTextLength() function gets the number of characters in inEntry’s text.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

Returns

Number of characters in inEntry’s text.

See also

sADMEntry->GetText()
sADMEntry->SetText()

sADMEntry->GetUserData()

Get the user data pointer for an entry

ADMUserData ASAPI (*GetUserData)(ADMEntryRef 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**

`sADMEntry->SetUserData()`

---

**`sADMEntry->Invalidate()`**

Invalidates the area of an entry

```c
void ASAPI (*Invalidate)(ADMEntryRef 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**

`sADMEntry->InvalidateRect()`
`sADMEntry->Update()`

---

**`sADMEntry->InvalidateRect()`**

Invalidates the area of a rectangle

```c
void ASAPI (*InvalidateRect)(ADMEntryRef 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**

<table>
<thead>
<tr>
<th>inEntry</th>
<th>An ADM entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inInvalRect</td>
<td>The area to be refreshed. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>
Returns
None.

See also
sADMEntry->Invalidate()
sADMEntry->Update()
sADMEntry->IsEnabled()  
Get whether or not an entry is enabled

ASBoolean ASAPI (*IsEnabled)(ADMEntryRef inEntry);

Description
The IsEnabled() function determines if inEntry is currently enabled. To change its state, use the sADMEntry->Enable() function. A disabled ADM entry is dimmed and unusable.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

Returns
true if inEntry is enabled; false otherwise.

See also
sADMEntry->Enable()

sADMEntry->IsInBounds()  
Find out whether an entry is visible

ASBoolean ASAPI (*IsInBounds)(ADMEntryRef inEntry);

Description
The IsInBounds() function determines whether inEntry is visible within the bounds of the list.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

Returns
true if inEntry is visible; false otherwise.

See also
sADMEntry->MakeInBounds()

sADMEntry->IsSelected()  
Get whether a list entry is selected

ASBoolean ASAPI (*IsSelected)(ADMEntryRef inEntry);
### Description

The `IsSelected()` function determines whether `inEntry` is currently selected. To change its state, use the `sADMEntry->Select()` function.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

### Returns

- `true` if `inEntry` is selected; `false` otherwise.

### See also

- `sADMEntry->Select()`
- `sADMEntry->GetIndex()`

---

### `sADMEntry->IsSeparator()`

**Description**

The `IsSeparator()` function determines whether `inEntry` is a separator. To change its state, use the `sADMEntry->MakeSeparator()` function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

**Returns**

- `true` if `inEntry` is a separator; `false` otherwise.

**See also**

- `sADMEntry->MakeSeparator()`

---

### `sADMEntry->LocalToScreenPoint()`

**Description**

The `LocalToScreenPoint()` function converts a point in `inEntry` to a point in the screen's coordinate space.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>ioPoint</code></td>
<td>A point in the screen's coordinate space.</td>
</tr>
</tbody>
</table>

**Returns**

- A point in the screen's coordinate space.

**See also**

- `sADMEntry->LocalToScreenPoint()`
Parameters

- **inEntry**: An ADM entry.
- **ioPoint**: A point in *inEntry*. Type: **ASPoint** (see **ASTypes.h**)

Returns

None.

See also

- `sADMEntry->LocalToScreenRect()`
- `sADMEntry->ScreenToLocalPoint()`

### `sADMEntry->LocalToScreenRect()`

Convert an entry rectangle to screen coordinates

```c
void ASAPI (*LocalToScreenRect)(ADMEntryRef 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 **ASTypes.h**)

**Returns**

None.

**See also**

- `sADMEntry->LocalToScreenPoint()`
- `sADMEntry->ScreenToLocalRect()`

### `sADMEntry->MakeInBounds()`

Make an entry visible

```c
void ASAPI (*MakeInBounds)(ADMEntryRef 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

sADMEntry->IsInBounds()

sADMEntry->MakeSeparator()

Set an entry to be a separator

void ASAPI (*MakeSeparator)(ADMEntryRef 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

sADMEntry->IsSeparator()

sADMEntry->ScreenToLocalPoint()

Convert a screen point to entry coordinates

void ASAPI (*ScreenToLocalPoint)(ADMEntryRef inEntry, ASPoint* ioPoint);

Description

The ScreenToLocalPoint() function converts a point in screen coordinates to a point relative to inEntry.
The ADM Entry Suite

ADM Entry Suite Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>ioPoint</td>
<td>A point in inEntry. Type: ASPoint (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMEntry->LocalToScreenPoint()
sADMEntry->ScreenToLocalRect()

sADMEntry->ScreenToLocalRect()

Convert a screen rectangle to entry coordinates

void ASAPI (*ScreenToLocalRect)(ADMEntryRef inEntry, ASRect* ioRect);

Description

The ScreenToLocalRect() function converts a rectangle in screen coordinates to a rectangle in the coordinate space of inEntry.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>ioRect</td>
<td>A rect in inEntry. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMEntry->ScreenToLocalPoint()
sADMEntry->LocalToScreenRect()

sADMEntry->Select()

Set whether a list entry is selected

void ASAPI (*Select)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>inSelect</td>
<td>Set to <code>true</code> to select <code>inEntry</code>; set to <code>false</code> to deselect it.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMEntry->IsSelected()`

`sADMEntry->SendNotify()`

Send a notification to an entry

```
void ASAPI (*SendNotify)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>inNotifierType</td>
<td>Notifier type.</td>
</tr>
</tbody>
</table>

Returns

None.

`sADMEntry->SetCheckGlyph()`

Set check glyph

```
void ASAPI (*SetCheckGlyph)(ADMEntryRef inEntry, ADMStandardCheckGlyphID inCheckGlyph);
```
The **SetCheckGlyph()** function sets the type if glyph for `inEntry`. Flyout menus can now have hyphens or bullets in addition to checkmarks.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>inCheckGlyph</code></td>
<td>The type of glyph to set for <code>inEntry</code>. Type: <code>ADMStandardCheckGlyphID</code> (see ADMEntry.h).</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

- `sADMEntry->GetCheckGlyph()`

---

**sADMEntry->SetDisabledPicture()**

Set the picture to display when entry is disabled

```c
void ASAPI (*SetDisabledPicture)(ADMEntryRef inEntry, ADMIconRef inPicture);
```

### Description

The **SetDisabledPicture()** function sets the picture that is used when `inEntry` is disabled.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>inPicture</code></td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

- `sADMEntry->GetDisabledPicture()`  
- `sADMEntry->SetDisabledPictureID()`  
- `sADMEntry->SetPicture()`  
- `sADMEntry->SetPictureID()`  
- `sADMEntry->SetSelectedPicture()`  
- `sADMEntry->SetSelectedPictureID()`
sADMEntry->SetDisabledPictureID()  
Set the entry’s disabled picture ID

void ASAPI (*SetDisabledPictureID)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>inPictureResID</td>
<td>Resource ID for the picture.</td>
</tr>
<tr>
<td>inPictureResName</td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMEntry->GetDisabledPictureID()  
sADMEntry->SetDisabledPicture()  
sADMEntry->SetPicture()  
sADMEntry->SetPictureID()   
sADMEntry->SetSelectedPicture()  
sADMEntry->SetSelectedPictureID()

sADMEntry->SetID()  
Set the ID of an entry

void ASAPI (*SetID)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>inEntryID</td>
<td>The ID of inEntry.</td>
</tr>
</tbody>
</table>

Adobe Dialog Manager Programmer’s Guide and Reference  289
Returns
None.

See also
sADMEntry->GetID()

sADMEntry->SetPicture()

Set the picture

```c
void ASAPI (*SetPicture)(ADMEntryRef inEntry, ADMIconRef inPicture);
```

Description
The `SetPicture()` function sets the picture to be displayed for `inEntry`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>inPicture</code></td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMEntry->GetPicture()
sADMEntry->SetPictureID()
sADMEntry->SetDisabledPicture()
sADMEntry->SetDisabledPictureID()
sADMEntry->SetSelectedPicture()
sADMEntry->SetSelectedPictureID()

sADMEntry->SetPictureID()

Set the entry's picture ID

```c
void ASAPI (*SetPictureID)(ADMEntryRef 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.
The ADM Entry Suite

ADM Entry Suite Functions

Parameters

<table>
<thead>
<tr>
<th>inEntry</th>
<th>An ADM entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPictureResID</td>
<td>Resource ID for the picture.</td>
</tr>
<tr>
<td>inPictureResName</td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMEntry->GetPictureID()

sADMEntry->SetPicture()

sADMEntry->SetDisabledPicture()

sADMEntry->SetDisabledPictureID()

sADMEntry->SetSelectedPicture()

sADMEntry->SetSelectedPictureID()

sADMEntry->GetSelectedPicture()

sADMEntry->SetPictureID()

sADMEntry->SetDisabledPicture()

sADMEntry->SetDisabledPictureID()

sADMEntry->SetSelectedPicture()

void ASAPI (*SetSelectedPicture)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>inEntry</th>
<th>An ADM entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPicture</td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMEntry->GetSelectedPicture()

sADMEntry->SetSelectedPictureID()

sADMEntry->SetPicture()

sADMEntry->SetPictureID()

sADMEntry->SetDisabledPicture()

sADMEntry->SetDisabledPictureID()
**sADMEntry->SetSelectedPictureID()**

Set the entry’s selected picture ID

```c
void ASAPI (*SetSelectedPictureID)(ADMEntryRef 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>inPictureResID</td>
<td>Resource ID for the picture.</td>
</tr>
<tr>
<td>inPictureResName</td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMEntry->GetSelectedPictureID()`
- `sADMEntry->SetSelectedPicture()`
- `sADMEntry->SetPicture()`
- `sADMEntry->SetPictureID()`
- `sADMEntry->SetDisabledPicture()`
- `sADMEntry->SetDisabledPictureID()`

---

**sADMEntry->SetText()**

Set the entry’s text

```c
void ASAPI (*SetText)(ADMEntryRef 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM entry.</td>
</tr>
<tr>
<td>inText</td>
<td>Text for <code>inEntry</code>.</td>
</tr>
</tbody>
</table>
Returns
None.

See also
sADMEntry->GetUserData()

sADMEntry->SetUserData()
Set the user data pointer for an entry

```c
void ASAPI (*SetUserData)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>inUserData</code></td>
<td>The 4-byte user value stored with <code>inEntry</code>. Type: <code>ADMUserData</code> (see <code>ADMTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMEntry->GetUserData()

sADMEntry->Update()
Force an update of an entry

```c
void ASAPI (*Update)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>
Returns

None.

See also

sADMEntry->Invalidate()
sADMEntry->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.

**sADMEntry->GetHelpID()**

Get the help ID of an entry

```
ASHelpID ASAPI (*GetHelpID)(ADMEntryRef 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**

sADMEntry->SetHelpID()

**sADMEntry->Help()**

Calls the help routine associated with an entry

```
void ASAPI (*Help)(ADMEntryRef inEntry);
```
Description

**NOTE:** This API is deprecated in ADM V2.8.

The `Help()` function calls the help for `inEntry`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
</tbody>
</table>

Returns

None.

```c
sADMEntry->SetHelpID()
```

Set the help ID

```c
void ASAPI (*SetHelpID)(ADMEntryRef 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM entry.</td>
</tr>
<tr>
<td><code>inHelpID</code></td>
<td>The resource ID for the ASHelp resource. Type: <strong>ASHelpID</strong> (See ASHelp.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMEntry->GetHelpID()`
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 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](image)

Adobe Dialog Manager Programmer’s Guide and Reference
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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

```c
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

```c
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.
The ADM Hierarchy List Suite

ADM Hierarchy List Suite Functions

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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

**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

```c
ASBoolean ASAPI (*GetDivided)(ADMHierarchyListRef inList);
```

**Description**

The `GetDivided()` function finds out whether `inList` is a divided list.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

**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.

**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 `sADMListEntry->DefaultDraw()` function.

**Parameters**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

**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: `ADMListEntryDrawProc` (see `ADMHierarchyList.h`)

sADMHierarchyList->GetEntry()

Get a list entry by ID

**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 `sADMListEntry->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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>inEntryID</td>
<td>ADM hierarchy list entry ID to search for.</td>
</tr>
</tbody>
</table>

**Returns**

The hierarchy list entry with the indicated ID. If no match is found, `NULL` is returned.
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>outRect</td>
<td>Location of the edit-in-place text item of <code>inList</code>. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>
Returns

None.

See also

sADMHierarchyList->SetEntryTextRect()
sADMHierarchyList->SetEntryTextRectRecursive()

\textbf{sADMHierarchyList->GetEntryWidth()} \hspace{1cm} \textbf{Get the width of a column of hierarchy list entries}

\begin{verbatim}
ASInt32 ASAPI (*GetEntryWidth)(ADMHierarchyListRef inList);
\end{verbatim}

Description

The \texttt{GetEntryWidth()} function returns the width of a column if \texttt{inList} is part of a listbox item and has the style \texttt{kADMTileListBoxStyle}. For any other list style, this function returns the width of the single column of the list.

Parameters

\begin{itemize}
  \item \texttt{inList} \hspace{1cm} An ADM hierarchy list.
\end{itemize}

Returns

Width of a column if \texttt{inList} is part of a listbox item and has the style \texttt{kADMTileListBoxStyle}; for any other list style, returns the width of the single column of the list.

\textbf{NOTE:} If the entry width has been set (using \texttt{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()

\textbf{sADMHierarchyList->GetExpandedIndex()} \hspace{1cm} \textbf{Get the expanded index}

\begin{verbatim}
ASInt32 ASAPI (*GetExpandedIndex)(ADMHierarchyListRef inList, ADMListEntryRef inEntry);
\end{verbatim}

Description

The \texttt{GetExpandedIndex()} returns the index of \texttt{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

<table>
<thead>
<tr>
<th>inList</th>
<th>An ADM hierarchy list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

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. |
The ADM Hierarchy List Suite

ADM Hierarchy List Suite Functions

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>
The initialization function used for the entries of \texttt{inList}. If you have not called \texttt{sADMHierarchyList->SetInitProc()}, returns \texttt{NULL} (not the default Init proc). Type: \texttt{ADMListEntryInitProc} (see \texttt{ADMHierarchyList.h})

See also

- \texttt{sADMHierarchyList->SetInitProc()}
- \texttt{sADMHierarchyList->SetInitProcRecursive()}

---

### \texttt{sADMHierarchyList->GetItem()}

Get the item reference for a hierarchy list

\[
\text{ADMItemRef ASAPI (*GetItem)(ADMHierarchyListRef inList);} \]

**Description**

The \texttt{GetItem()} function returns a reference to the ADM item to which \texttt{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**

- \texttt{inList} An ADM hierarchy list.

**Returns**

The ADM item to which \texttt{inList} belongs.

---

### \texttt{sADMHierarchyList->GetLeafIndex()}

Get index of entry

\[
\text{ASInt32 ASAPI (*GetLeafIndex)(ADMHierarchyListRef inList, ADMListEntryRef inEntry);} \]

**Description**

The \texttt{GetLeafIndex()} function gets the index of \texttt{ADMListEntryRef} in \texttt{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**

- \texttt{inList} An ADM hierarchy list.
- \texttt{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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>outRect</td>
<td>inList's local rectangle. Type: outRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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()
The ADM Hierarchy List Suite

ADM Hierarchy List Suite Functions

`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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

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

```c
ADMLListEntryRef ASAPI (*GetParentEntry)(ADMHierarchyListRef inList);
```

**Description**

The `GetParentEntry()` function returns the parent of `inList`. Returns **NULL** if `inList` is a root level hierarchy list.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

**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

```c
ADMLListEntryTrackProc 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 `sADMLListEntry->DefaultTrack()` function.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

**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: `ADMLListEntryTrackProc` (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

<table>
<thead>
<tr>
<th>inList</th>
<th>An ADM hierarchy list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioRect</td>
<td>The rectangle converted from “global” to local space. Type: outRect (See ASTypes.h)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>inSelectionIndex</td>
<td>Index of the entry in inList to return a reference to.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>inIndex</td>
<td>Index of the entry in inList to return a reference to.</td>
</tr>
</tbody>
</table>

Returns

The ADM list entry at index inIndex.
Example

```c
ADMHierarchyListRef list;
ADMListEntryRef entry;
// assign value to list variable
ASInt32 i, count = sADMHierarchyList-&gt;NumberOfEntries(list);

for (i = 0; i &lt; count; i++) {
    entry = sADMHierarchyList-&gt;IndexEntry(list, i);
    // do something with the entry
}
```

### sADMHierarchyList-&gt;IndexExpandedEntry()

**Get an expanded list entry by index**

```c
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-&gt;NumberOfExpandedEntriesInHierarchy()` to retrieve references to all selected entries.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inExpandedItem</code></td>
<td>Index of the expanded list entry to retrieve.</td>
</tr>
</tbody>
</table>

**Returns**

The ADM list entry at index `inExpandedItem`.

**See also**

`sADMHierarchyList-&gt;NumberOfExpandedEntriesInHierarchy()`

### sADMHierarchyList-&gt;IndexLeafEntry()

**Get the entry at the specified index**

```c
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 \textit{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**

- \textit{inList} An ADM hierarchy list.
- \textit{inLeafItem} Index of the leaf entry in \textit{inList} to return a reference to.

**Returns**

The ADM list entry at index \textit{inLeafItem}.

**See also**

- \texttt{sADMHierarchyList->NumberOfLeafEntries()}

---

\textbf{sADMHierarchyList->IndexSelectedEntry()}

Get one of multiple selected list entries

\begin{verbatim}
ADMListEntryRef ASAPI
(*IndexSelectedEntry)(ADMHierarchyListRef inList, ASInt32 inSelectionIndex);
\end{verbatim}

**Description**

The \texttt{IndexSelectedEntry()} function gets a reference to any one of several selected entries of \textit{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 \texttt{sADMHierarchyList->NumberOfSelectedEntries()}, you can iterate through all of a hierarchy list’s selected entries (see example).

**Parameters**

- \textit{inList} An ADM hierarchy list.
- \textit{inSelectionIndex} Index of the entry in \textit{inList} to return a reference to.

**Returns**

The ADM list entry at index \textit{inSelectionIndex}.  

---
Example

```c
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

```c
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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inIndex</code></td>
<td>Zero-based index of where to add the new ADM list entry.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inEntry</code></td>
<td>The new ADM list entry to add.</td>
</tr>
<tr>
<td><code>inIndex</code></td>
<td>Index of where to add the new ADM list entry.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- `sADMHierarchyList->StartMultipleItemInvalidate()`
- `sADMHierarchyList->StopMultipleItemInvalidate()`

---

**sADMHierarchyList->LocalToGlobalPoint()**

This function converts a point to a global coordinate.

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><strong>ioPoint</strong></td>
<td>The point converted from local to “global” space. Type: <code>ASPoint</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- `sADMHierarchyList->GlobalToLocalPoint()`
- `sADMHierarchyList->LocalToScreenPoint()`
- `sADMHierarchyList->LocalToGlobalRect()`
sADMHierarchyList->LocalToGlobalRect()

Convert a local rectangle to global rectangle

```c
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**

<table>
<thead>
<tr>
<th>inList</th>
<th>An ADM hierarchy list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioRect</td>
<td>The rect converted from local to “global” space. Type: outRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMHierarchyList->GlobalToLocalRect()` 
- `sADMHierarchyList->LocalToGlobalPoint()` 
- `sADMHierarchyList->LocalToGlobalRect()`

---

sADMHierarchyList->LocalToScreenPoint()

Convert a point to a screen coordinate

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>ioPoint</code></td>
<td>The point in <code>inList</code> converted to a point in the screen coordinate space. Type: <code>ASPoint</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

- `sADMHierarchyList->ScreenToLocalPoint()`
- `sADMHierarchyList->LocalToGlobalPoint()`
- `sADMHierarchyList->LocalToGlobalRect()`

#### `sADMHierarchyList->NumberOfEntries()` Get the number of list entries in a hierarchy list

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

### Returns

The number of entries in `inList`.

### See also

- `sADMHierarchyList->IndexEntry()`

#### `sADMHierarchyList->NumberOfLeafEntries()` Get the number of leaf entries

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

### 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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

### Returns

The number of selected root-level entries in **inList**.

### See also

- **sADMHierarchyList->IndexSelectedEntry()**
The ADM Hierarchy List Suite

ADM Hierarchy List Suite Functions

---

**sADMHierarchyList->NumberOfAllSelectedEntriesInHierarchy()**

Get the number of all selected list entries in a hierarchy list

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

**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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

**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.

```c
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()`

**sADMHierarchyList->PickEntry()**

Select a list entry at a specified point

```c
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()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>inPoint</td>
<td>Location of inList entry to be selected. Type: ASPoint (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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.
The ADM Hierarchy List Suite

ADM Hierarchy List Suite Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>inIndex</td>
<td>Index of entry to remove.</td>
</tr>
</tbody>
</table>

Returns

None.

sADMHierarchyList->ScreenToLocalPoint() Convert a screen point to list coordinates

```c
void ASAPI (*ScreenToLocalPoint)(ADMHierarchyListRef inList, ASPoint* ioPoint);
```

Description

The `ScreenToLocalPoint` function converts a point in screen coordinates to a point relative to `inList`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>ioPoint</td>
<td>The point in <code>inList</code> in screen coordinates that is converted to a point relative to <code>inList</code>. Type: <code>ASPoint</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMHierarchyList->LocalToScreenPoint()
sADMHierarchyList->LocalToGlobalPoint()
sADMHierarchyList->LocalToGlobalRect()

sADMHierarchyList->SetBackgroundColor() Set the background color

```c
void ASAPI (*SetBackgroundColor)(ADMHierarchyListRef inList, ADMColor inColor);
```

Description

The `SetBackgroundColor` function sets the background color.
### Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inColor</td>
<td>Type: ADMColor (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

### Returns

None.

### `sADMHierarchyList->SetDestroyProc()`

**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

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDestroyProc</td>
<td>A callback with the following signature (see ADMHierarchyList.h):</td>
</tr>
<tr>
<td></td>
<td><code>ADMListEntryDestroyProc(ADMListEntryRef inEntry);</code></td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

- `sADMHierarchyList->GetDestroyProc()`
- `sADMHierarchyList->SetDestroyProcRecursive()`

### `sADMHierarchyList->SetDestroyProcRecursive()`

**Set the ADM destroy function for the hierarchy list’s entries**

```c
void ASAPI (*SetDestroyProc)(ADMHierarchyListRef inList,
ADMListEntryDestroyProc inDestroyProc);
```
The ADM Hierarchy List Suite

ADM Hierarchy List Suite Functions

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>
| `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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>
### Returns
None.

### See also
`sADMHierarchyList->SetDividedRecursive()`

### `sADMHierarchyList->SetDividedRecursive()`

Make a list and all its children into ADM divided lists

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inDivided</code></td>
<td>If <code>true</code>, makes <code>inList</code> and the lists of children of the list’s entries divided; if <code>false</code>, <code>inList</code> and the lists of children of the list’s entries are not divided.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**See also**
`sADMHierarchyList->SetDivided()`

### `sADMHierarchyList->SetDrawProc()`

Set the ADM drawing function to use for the hierarchy list’s entries

```c
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 `sADMListEntry- >DefaultDraw()` function.

See Using Event Callbacks in Chapter 1, “ADM Overview” for more information.

**Parameters**

<table>
<thead>
<tr>
<th>inList</th>
<th>An ADM hierarchy list.</th>
</tr>
</thead>
</table>
| inDrawProc | A callback with the following signature (see ADMHierarchyList.h):

```c
ADMListEntryDrawProc(ADMListEntryRef inEntry, ADMDrawerRef inDrawer);
```

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.

**Returns**

None.

**See also**

- `sADMHierarchyList->GetDrawProc()`
- `sADMHierarchyList->SetDrawProcRecursive()`

---

**sADMHierarchyList->SetDrawProcRecursive()**

```c
void ASAPI (*SetDrawProcRecursive)(ADMHierarchyListRef inList,
ADMListEntryDrawProc 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 `sADMListEntry- >DefaultDraw()` function.

See Using Event Callbacks in Chapter 1, “ADM Overview” for more information.

**Parameters**

| inList | An ADM hierarchy list. |
### ADM Hierarchy List Suite Functions

#### sADMHierarchyList->SetEntryHeight()

Set the height of a hierarchy list's entries

```c
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()`

---

#### inDrawProc

A callback with the following signature (see `ADMHierarchyList.h`):

```
ADMListEntryDrawProc(ADMListEntryRef inEntry, ADMDrawerRef inDrawer);
```

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.

**Returns**

None.

**See also**

- `sADMHierarchyList->GetDrawProc()`
- `sADMHierarchyList->SetDrawProcRecursive()`

---
sADMHierarchyList->SetEntryHeightRecursive()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>inHeight</td>
<td>The height of a hierarchy list's entry.</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMHierarchyList->GetEntryHeight()
sADMHierarchyList->SetEntryHeight()

sADMHierarchyList->SetEntryTextRect()

void ASAPI (*SetEntryTextRect)(ADMHierarchyListRef inList, const ASRect* inRect);

Description
The SetEntryTextRect() function sets the edit-in-place text item of a hierarchy list.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>inRect</td>
<td>Location of the edit-in-place text item of inList. Type: outRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns
None.
See also

`sADMHierarchyList->GetEntryTextRect()`
`sADMHierarchyList->SetEntryTextRectRecursive()`

---

**sADMHierarchyList->SetEntryTextRectRecursive()**

Recursively set the edit text rectangle for a hierarchy list.

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>Location of the edit-in-place text items of <code>inList</code> and any children. Type: <code>outRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMHierarchyList->GetEntryTextRect()`
`sADMHierarchyList->SetEntryTextRect()`

---

**sADMHierarchyList->SetEntryWidth()**

Set the width of a column of list entries

```c
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.
The ADM Hierarchy List Suite

ADM Hierarchy List Suite Functions

Parameters

<table>
<thead>
<tr>
<th>inList</th>
<th>An ADM hierarchy list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inWidth</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMHierarchyList->GetEntryWidth()
sADMHierarchyList->SetEntryWidthRecursive()

sADMHierarchyList->SetEntryWidthRecursive()

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

<table>
<thead>
<tr>
<th>inList</th>
<th>An ADM hierarchy list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inWidth</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMHierarchyList->GetEntryWidth()
sADMHierarchyList->SetEntryWidth()
sADMHierarchyList->SetFlags()

Set the flags associated with the list.

```c
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`.

```c
// 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 `ADMHierarchyList.h`.

**Returns**

None.

**See also**

- `sADMHierarchyList->GetFlags()`
- `sADMHierarchyList->SetFlagsRecursive()`

sADMHierarchyList->SetFlagsRecursive()

Set the flags associated with the list and its children.

```c
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 ADMHierarchyList.h.

### Returns
None.

### See also
- sADMHierarchyList->GetFlags()
- sADMHierarchyList->SetFlags()

### sADMHierarchyList->SetIndentationWidth()

**Set the spacing for the expand/collapse arrow**

```c
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.
sADMHierarchyList->SetIndentationWidthRecursive()

Set the spacing for the expand/collapse arrow for a list and its children

```c
void ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inWidth</code></td>
<td>Indentation spacing for the expand/collapse arrow for <code>inList</code>.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMHierarchyList->GetIndentationWidth()`
- `sADMHierarchyList->SetIndentationWidthRecursive()`

sADMHierarchyList->SetInitProc()

Set the ADM init function to use for hierarchy list entries

```c
void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>inList</em></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>
| *inInitProc*| A callback with the following signature (see [ADMHierarchyList.h](#)):  

```c
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**

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>inList</em></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>
The ADM Hierarchy List Suite

ADM Hierarchy List Suite Functions

### inInitProc

A callback with the following signature (see ADMHierarchyList.h):

```c
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)

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>inWidth</td>
<td>Width of left margin, in pixels.</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

- sADMHierarchyList->GetLocalLeftMargin()

---

### sADMHierarchyList->SetMask()

Set a filter on events received by the hierarchy list entries' tracker

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><strong>inMask</strong></td>
<td>The action mask for <strong>inList</strong>. Maskable actions are listed in ADMTracker.h. Type: <strong>ADMActionMask</strong> (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- sADMHierarchyList->GetMask()
- sADMHierarchyList->SetMaskRecursive()

---

**sADMHierarchyList->SetMaskRecursive()**

Set a filter on events received by the hierarchy list entries’ tracker

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><strong>inMask</strong></td>
<td>The action mask for <strong>inList</strong>. Maskable actions are listed in ADMTracker.h. Type: <strong>ADMActionMask</strong> (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- sADMHierarchyList->GetMask()
- sADMHierarchyList->SetMask()
sADMHierarchyList->SetMenuID()  
Set the menu resource ID of a hierarchy list

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inMenuResID</code></td>
<td>Resource ID for <code>inList</code>. ID of standard platform menu resource.</td>
</tr>
<tr>
<td><code>inMenuResName</code></td>
<td>List resource name.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMHierarchyList->GetMenuID()`

---

sADMHierarchyList->SetNonLeafEntryTextRect()  
Set the non-leaf rectangle for a hierarchy list

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>
Returns
None.

See also
sADMHierarchyList->GetNonLeafEntryTextRectRecursive()

sADMHierarchyList->SetNonLeafEntryTextRectRecursive()
Set the non-leaf rectangle for a hierarchy list and any children

void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td>inRect</td>
<td>The non-leaf text rectangle for inList and the lists of children of inList’s entries. Type: outRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMHierarchyList->GetNonLeafEntryWidth()

sADMHierarchyList->SetNotifierData()
Set notifier data for a hierarchy list entry

void ASAPI (*SetNotifierData)(ADMHierarchyListRef inEntry, ADMUserData inData);

Description
The SetNotifierData() function sets the notification data for inList, if any. This is used for custom notification procedures.
### sADMHierarchyList->SetNotifyProc()

Set the ADM notification function to use for the hierarchy list's entries.

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inNotifyProc</code></td>
<td>A callback with the following signature (see ADMHierarchyList.h):</td>
</tr>
</tbody>
</table>

```c
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.
The ADM Hierarchy List Suite

ADM Hierarchy List Suite Functions

Returns

None.

See also

sADMHierarchyList->SetNotifyProcRecursive()
sADMHierarchyList->GetNotifyProc()

sADMHierarchyList->SetNotifyProcRecursive()

Set the ADM notification function for the hierarchy list's entries and children

void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>
| inNotifyProc      | A callback with the following signature (see ADMHierarchyList.h):
|                   | `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->SetTrackProc()

void ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inTrackProc</code></td>
<td>A callback with the following signature (see <code>ADMHierarchyList.h</code>):</td>
</tr>
<tr>
<td></td>
<td><code>ASBoolean ADMListEntryTrackProc (ADMListEntryRef inEntry, ADMTrackerRef inTracker);</code></td>
</tr>
<tr>
<td></td>
<td><code>inTracker</code> is passed to the ADM Tracker suite functions to indicate the event for which information is being requested. This callback returns a boolean—if <code>true</code>, the item receives a notify event when the mouse is released; if <code>false</code>, a notify event will not be received. It always means whether or not to send notification except for keystroke events. In those cases it means whether or not the keystroke was handled.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMHierarchyList->GetTrackProc()`
- `sADMHierarchyList->SetTrackProcRecursive()`

sADMHierarchyList->SetTrackProcRecursive()

Set the ADM tracker function to use for the hierarchy list's entries...
void ASAPI (*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 ADMHierarchyList.h):

\[
\text{ADMListEntryTrackProc}(\text{ADMListEntryRef inEntry, ADMTrackerRef inTracker});
\]

**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 except 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 ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inData</code></td>
<td>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 <code>sADMHierarchyList-&gt;GetUserData()</code> function. Type: <code>ADMUserData</code> (see <code>ADMTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMHierarchyList->GetUserData()`

`sADMHierarchyList->StartMultipleItemInvalidate()`

Start invalidation.

```c
void ASAPI (*StartMultipleItemInvalidate)(ADMHierarchyListRef inList);
```

Description

The `StartMultipleItemInvalidate()` function allows you to limit invalidation to improve performance. Use with `sADMHierarchyList->StopMultipleItemInvalidate()`.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMHierarchyList->StopMultipleItemInvalidate()`
`sADMHierarchyList->Invalidate()`

`sADMHierarchyList->StopMultipleItemInvalidate()`

Stop invalidation

```c
void ASAPI (*StopMultipleItemInvalidate)(ADMHierarchyListRef inList);
```
Description
The **StartMultipleItemInvalidate()** function allows you to limit invalidation to improve performance. Use with **sADMHierarchyList->StartMultipleItemInvalidate()**.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><strong>inFromIndex</strong></td>
<td>Entry to be swapper with <strong>inToIndex</strong>.</td>
</tr>
<tr>
<td><strong>inToIndex</strong></td>
<td>Entry to be swapper with <strong>inFromIndex</strong>.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**See also**
- **sADMHierarchyList->StartMultipleItemInvalidate()**
- **sADMHierarchyList->Invalidate()**

**sADMHierarchyList->SwapEntries()**

```
void ASAPI (*SwapEntries)(ADMHierarchyListRef inList, ASInt32 inFromIndex, ASInt32 inToIndex);
```

Description
The **SwapEntries()** function switches the position of two entries.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><strong>inFromIndex</strong></td>
<td>Entry to be swapper with <strong>inToIndex</strong>.</td>
</tr>
<tr>
<td><strong>inToIndex</strong></td>
<td>Entry to be swapper with <strong>inFromIndex</strong>.</td>
</tr>
</tbody>
</table>

**Returns**
None.

**sADMHierarchyList->UnlinkEntry()**

```
ADMListEntryRef ASAPI (*UnlinkEntry)(ADMHierarchyListRef inList, ASInt32 inIndex);
```

Unlink an entry from a hierarchy list.
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM hierarchy list.</td>
</tr>
<tr>
<td><code>inIndex</code></td>
<td>Index of the list entry to be unlinked.</td>
</tr>
</tbody>
</table>

Returns

Unlinked ADM list entry at `inIndex`. 
The ADM Hierarchy List Suite
ADM Hierarchy List Suite Functions
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:

```c
#define kADMIconSuite "ADM Icon Suite"
```

with the version constant:

```c
#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:

```c
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`):

```c
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. \texttt{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 \texttt{kIconSuite} and \texttt{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)

\textbf{NOTE:} On Macintosh, the \texttt{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 \texttt{sADMDrawer->DrawIcon()} and \texttt{sADMDrawer->DrawIconCentered()}.

---

**ADM Icon Suite Functions**

**sADMIIcon->Create()**

Create a new icon

\begin{verbatim}
ADMIconRef ASAPI (*Create)(ADMIconType inIconType, ASInt32 inWidth, ASInt32 inHeight, void* inData);
\end{verbatim}

**Description**

The \texttt{Create()} function creates a new icon of type \texttt{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 \texttt{sADMIIcon->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**

<table>
<thead>
<tr>
<th>inIconType</th>
<th>Icon type. Type: \texttt{ADMIconType} (see \texttt{ADMIcon.h})</th>
</tr>
</thead>
<tbody>
<tr>
<td>inWidth</td>
<td>Width of icon (in pixels).</td>
</tr>
<tr>
<td>inHeight</td>
<td>Height of icon (in pixels).</td>
</tr>
</tbody>
</table>
### 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

```c
ADMIconRef ASAPI (*CreateFromImage)(ADMImageRef inImage);
```

### Description
The `CreateFromImage()` function enables ADM icon to take ownership of the image.

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inImage</td>
<td>An ADM image.</td>
</tr>
</tbody>
</table>

### Returns
A new ADM icon that now owns `inImage`.

### See also
- `sADMIcon->Create()`
- `sADMIcon->Destroy()`

### sADMIcon->Destroy()
Remove an icon from memory

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inIcon</td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>
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

ADMIIconRef 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

<table>
<thead>
<tr>
<th>inPluginRef</th>
<th>Plug-in reference.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inName</td>
<td>Internal identifier for the icon. Must be unique.</td>
</tr>
<tr>
<td>inIconID</td>
<td>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.</td>
</tr>
<tr>
<td>inIconIndex</td>
<td>Index added to inIconID to get the final resource ID of the icon.</td>
</tr>
</tbody>
</table>

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()`
- `sADMIcon->GetHeight()`
- `sADMIcon->IsFromResource()`
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inIcon</td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>

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()
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:

```c
ADMImageSuite *sADMImage;
error = sSPBasic->AcquireSuite(kADMImageSuite, kADMImageSuiteVersion2, &sADMImage);
if (error) . . . //handle error
```

For SuitePea errors, see `SPErrorCodes.h`.

ADM Image Suite Functions

### `sADMIage->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 `sADMIage->EndADMDrawer()`.
The ADM Image Suite

ADM Image Suite Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inImage</td>
<td>An ADM image.</td>
</tr>
</tbody>
</table>

Returns

An ADM drawer.

See also

sADMImage->EndADMDrawer()

s ADMImage->BeginAGMImageAccess()

Begin using AGM routines to draw an image

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inImage</td>
<td>An ADM image.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>outImageRecord</td>
<td>The structure that describes <code>inImage</code>. Type: <code>_t_ADMAGMImageRecord</code> (see <code>ADMAGMTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMImage->EndBaseAddressAccess()
sADMImage->GetPixel()

sADMImage->BeginBaseAddressAccess()

Get the starting address of an image in memory

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inImage</strong></td>
<td>An ADM image.</td>
</tr>
</tbody>
</table>

Returns

Pointer to the beginning address of **inImage** in memory. Type: **ASBytePtr** (see **ASTypes.h**)

See also

**sADMImage->EndBaseAddressAccess()**

### **sADMImage->Create()**

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inWidth</strong></td>
<td>Width of the new image (in pixels).</td>
</tr>
<tr>
<td><strong>inHeight</strong></td>
<td>Height of the new image (in pixels).</td>
</tr>
<tr>
<td><strong>inOptions</strong></td>
<td>Pass 0 (zero) or <strong>kADMImageHasAlphaChannelOption</strong> (1L &lt;&lt; 0).</td>
</tr>
</tbody>
</table>

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

```c
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

```c
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.
The ADM Image Suite

ADM Image Suite Functions

See also

`sADMImage->BeginAGMImageAccess()`

---

`sADMImage->EndBaseAddressAccess()`  Find out the end address of an image in memory

```c
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

```c
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. ((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. |
The ADM Image Suite

ADM Image Suite Functions

Returns

Height of \texttt{inImage}.

See also

\texttt{sADMImage}\rightarrow\texttt{GetWidth()}

---

\texttt{sADMImage}\rightarrow\texttt{GetPixel()}

Get pixel data at a given point

\begin{verbatim}
ASErr ASAPI (*GetPixel)(ADMImageRef inImage, const ASPoint* inPoint, ASRGBColor* outColor);
\end{verbatim}

Description

The \texttt{GetPixel()} function obtains \texttt{outColor} for the pixel at \texttt{inPoint}. If you are working with all the pixels of an image, you may want to use the \texttt{sADMImage}\rightarrow\texttt{BeginBaseAddressAccess()} and \texttt{sADMImage}\rightarrow\texttt{EndBaseAddressAccess()} routines for speed.

Parameters

\begin{verbatim}
inImage An ADM image.
inPoint The pixel for which to obtain color data. Type: ASPoint (see ASTypes.h)
outColor The color of pixel at \texttt{inPoint}. Type: ASRGBColor (see ASTypes.h)
\end{verbatim}

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred.

See also

\texttt{sADMImage}\rightarrow\texttt{SetPixel()}
\texttt{sADMImage}\rightarrow\texttt{BeginBaseAddressAccess()}
\texttt{sADMImage}\rightarrow\texttt{EndBaseAddressAccess()}

---

\texttt{sADMImage}\rightarrow\texttt{GetWidth()}

Get the width of an image

\begin{verbatim}
ASInt32 ASAPI (*GetWidth)(ADMImageRef inImage)
\end{verbatim}

The \texttt{GetWidth()} function gets the width of \texttt{inImage}.

Parameters

\begin{verbatim}
inImage An ADM image.
\end{verbatim}
Returns

Width of \texttt{inImage} (in pixels).

See also

\texttt{sADMImage->GetHeight()}

---

\textbf{\texttt{sADMImage->SetPixel()}}

Set pixel data at a given point

\[
\text{ASErr ASAPI (*SetPixel)(ADMImageRef inImage, const ASPoint* inPoint, const ASRGBColor* inColor);}
\]

Description

The \texttt{SetPixel()} function sets the color data (\texttt{inColor}) for the pixel at \texttt{inPoint}.

Parameters

\begin{tabular}{ | l | l |}
\hline
\textbf{\texttt{inImage}} & An ADM image. \\
\hline
\textbf{\texttt{inPoint}} & The pixel for which to set color data. Type: \texttt{ASPoint} (see \texttt{ASTypes.h}) \\
\hline
\textbf{\texttt{inColor}} & The color for pixel at \texttt{inPoint}. Type: \texttt{ASRGBColor} (see \texttt{ASTypes.h}) \\
\hline
\end{tabular}

Returns

0 if operation was successful; otherwise, the error code indicates the error that occurred.

See also

\texttt{sADMImage->GetPixel()}
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:

```c
#define kADMItemSuite "ADM Item Suite"
```

with the version constant:

```c
#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:

```c
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

<table>
<thead>
<tr>
<th>Item Type (see ADMItem.h)</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>kADMDialType</td>
<td>Set the dial’s minimum, maximum, and current value. Assign a notification proc.</td>
</tr>
<tr>
<td>kADMFrameType</td>
<td>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.</td>
</tr>
</tbody>
</table>
| kADMItemGroupType                      | Nothing needs to be done in an item group’s InitProc. At the time the Init 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:  
  1. `sADMItem->Create()` the item group (this causes the InitProc() to be called prior to return from `sADMItem->Create()`)
  2. Add any child items that need to be added to this group (use `sADMItem->AddItem()`)
  3. Set any properties for the group that should be propagated to the children (using the various ADM Item suite Set mutators).
| kADMTabbedMenuType                     | Obsolete.                                                                                                                                                                                                     |
| kADMListBoxType                        | Get the item’s list and set its menu ID. If necessary, activate an entry or entries. Assign a notification function for the entries.                                                                             |
| kADMHierarchyListBoxType               | 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.                                                     |
| kADMPicturePushButtonType              | Set the picture ID. You may want to disable and select picture IDs as well. Also, a push button needs a notification function.                                                                             |
| kADMPictureCheckBoxType                | 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

<table>
<thead>
<tr>
<th>Item Type (see ADMItem.h)</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>kADMPictureRadioButtonType</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>kADMPictureStaticType</code></td>
<td>Set the picture ID. You may want to disable and select picture IDs as well. You may want a notification for an easter egg.</td>
</tr>
<tr>
<td><code>kADMPopupControlType</code></td>
<td>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 <code>Set***Value()</code> functions.</td>
</tr>
<tr>
<td><code>kADMPopupButtonType</code></td>
<td>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 <code>Set***Value()</code> functions.</td>
</tr>
<tr>
<td><code>kADMPopupSpinEditControlType</code></td>
<td>Set the item's text. Note that these types of items can have only numeric data (floats or ints, negative values are acceptable). Use <code>sADMItem-&gt;SetText()</code> or <code>Set***Value()</code> 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.</td>
</tr>
<tr>
<td><code>kADMPopupListType</code></td>
<td>Get the item's list and set its menu ID. If necessary, activate an entry or entries.</td>
</tr>
<tr>
<td><code>kADMPopupMenuType</code></td>
<td>Get the item's list and set its menu ID. If necessary, activate an entry or entries.</td>
</tr>
<tr>
<td><code>kADMScrollingPopupListType</code></td>
<td>Get the item's list and set its menu ID. If necessary, activate an entry or entries.</td>
</tr>
<tr>
<td><code>kADMResizeType</code></td>
<td>Set a notification function for the item.</td>
</tr>
<tr>
<td><code>kADMScrollbarType</code></td>
<td>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).</td>
</tr>
</tbody>
</table>
### Initialization of ADM Items

<table>
<thead>
<tr>
<th>Item Type (see ADMItem.h)</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>kADMSliderType</td>
<td>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).</td>
</tr>
<tr>
<td>kADMSpinEditType</td>
<td>Set the item's text. Set any numerics, including ranges, units, etc. Set the increments to be used for the spinner.</td>
</tr>
<tr>
<td>kADMSpinEditPopupType</td>
<td>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.</td>
</tr>
<tr>
<td>kADMSpinEditScrollingPopupType</td>
<td>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.</td>
</tr>
<tr>
<td>kADMTextEditType</td>
<td>Set the item's text and maximum text length. Set any numerics, including ranges, units, and precision.</td>
</tr>
<tr>
<td>kADMTextEditReadOnlyType</td>
<td>Same as kADMTextEditType except that you can't type in them and you can't select them.</td>
</tr>
<tr>
<td>kADMTextEditMultiLineType</td>
<td>Set the item's text and maximum text length. Set any numerics, including ranges, units, and precision.</td>
</tr>
<tr>
<td>kADMTextEditMultiLineReadOnlyType</td>
<td>Same as kADMTextEditMultiLineType except you can't type in them and you can't select them.</td>
</tr>
<tr>
<td>kADMTextEditPopupType</td>
<td>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.</td>
</tr>
<tr>
<td>kADMTextEditScrollingPopupType</td>
<td>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.</td>
</tr>
<tr>
<td>ADMTextPushButtonType</td>
<td>Push buttons need a notification routine.</td>
</tr>
</tbody>
</table>
The ADM Item Suite

About the ADM Item Suite

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

---

<table>
<thead>
<tr>
<th>Item Type (see <code>ADMItem.h</code>)</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ADMTextRadioButtonType</code></td>
<td>Set the boolean value of one radio button in a group to be <code>true</code>. 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.</td>
</tr>
<tr>
<td><code>ADMTextStaticType</code></td>
<td>Possibly enable or disable.</td>
</tr>
<tr>
<td><code>kADMTextStaticMultilineType</code></td>
<td>Possibly enable or disable.</td>
</tr>
<tr>
<td><code>kADMProgressBarType</code></td>
<td>Set the initial value. This item type typically exists transiently while an operation is being performed.</td>
</tr>
<tr>
<td><code>kADMChasingArrowsType</code></td>
<td>Only implemented on the Mac. This item would typically be used transiently to indicate that the user needs to wait. No initialization required.</td>
</tr>
<tr>
<td><code>ADMUserType</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>kADMPasswordTextEditType</code></td>
<td>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).</td>
</tr>
</tbody>
</table>

---

Adobe Dialog Manager Programmer’s Guide and Reference 379
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

```c
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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inActivate</code></td>
<td>If <code>true</code>, <code>inItem</code> is activated; if <code>false</code>, it is inactivated.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMItem->IsActive()`

**sADMItem->AddItem()**  
Add an item to a group

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inGroup</code></td>
<td>An ADM item group.</td>
</tr>
</tbody>
</table>
**Create()**

Create a new ADM item

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
<tr>
<td><code>inItemID</code></td>
<td>Reference ID of the object. When adding an item to an existing dialog, use the <code>kADMUniqueItemID</code> constant and ADM generates an ID for you.</td>
</tr>
<tr>
<td><code>inItemType</code></td>
<td>Item type. Type: <code>ADMItemType</code> (see ADMItem.h)</td>
</tr>
<tr>
<td><code>inBoundsRect</code></td>
<td>Position and size of the item within <code>inDialog</code>'s coordinate space.</td>
</tr>
</tbody>
</table>
The ADM Item Suite

ADM Item Suite Functions

## ADM Item Suite Functions

### `CreateTimer()`

Create a timer

```c
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.
The ADM Item Suite

**ADM Item Suite Functions**

### Returns

None.

### See also

- `sADMItem->AbortTimer()`

---

#### `sADMItem->DefaultDraw()`

**Call ADM's default item draw function**

```c
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.

---

<table>
<thead>
<tr>
<th><strong>inAbortMask</strong></th>
<th>Specifies actions that result in calling <code>inTimerAbortProc</code>. Type: <code>ADMActionMask</code> (see <code>ADMTypes.h</code>)</th>
</tr>
</thead>
</table>
| **inTimerProc** | Callback with the following signature (see `ADMItem.h`):
  ```c
  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`):
  ```c
  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). |
Parameters

<table>
<thead>
<tr>
<th>inItem</th>
<th>An ADM item.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMItem->SetDrawProc()

Example

```c
void doNothingDrawHandler(ADMItemRef inItem, ADMDrawerRef inDrawer) {
    sADMItem->DefaultDraw(inItem, inDrawer);
}
```

sADMItem->DefaultFloatToText()  

ADM's default procedure for converting a floating point number to text

```c
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

<table>
<thead>
<tr>
<th>inItem</th>
<th>An ADM item.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inValue</td>
<td>Floating point value for conversion.</td>
</tr>
<tr>
<td>outText</td>
<td>inValue converted to text.</td>
</tr>
<tr>
<td>inMaxLength</td>
<td>Size of outText buffer.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMItem->SetFloatToTextProc()
sADMItem->DefaultTextToFloat()  
ADM's default procedure for converting a text string to a floating point number

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inText</code></td>
<td>Text for conversion.</td>
</tr>
<tr>
<td><code>outValue</code></td>
<td><code>inText</code> converted to a floating point number.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

sADMItem->SetTextToFloatProc()

sADMItem->DefaultNotify()  
Call an ADM item's default notification function

```c
void ASAPI (*DefaultNotify)(ADMItemRef inItem, ADMMonitorRef 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>AN ADM item.</td>
</tr>
<tr>
<td><code>inNotifier</code></td>
<td>An ADM notifier.</td>
</tr>
</tbody>
</table>
Returns

None.

See also

sADMItem->SetNotifyProc()

Example

```c
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inTracker</code></td>
<td>An ADM tracker.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inEnable</code></td>
<td>If <code>true</code>, <code>inItem</code> is enabled; if <code>false</code>, <code>inItem</code> is disabled.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMItem->isEnabled()`

---

`sADMItem->GetAllowMath()`

Gets whether an item uses math in computing numeric values

```c
ASBoolean ASAPI (*GetAllowMath)(ADMItemRef inItem);
```

Description

The **GetAllowMath()** function determines whether math can be used in computing user input values for an item.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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

```c
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

```c
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

```c
void ASAPI (*GetBestSize)(ADMItemRef inItem, ASprite* 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.
The ADM Item Suite

ADM Item Suite Functions

**sADMItem->GetBooleanValue()**

Get the boolean value of an item

```c
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

```c
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.

```c
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->GetChildIndex()
Get a child item of an item

ADMItemRef ASAPI (*GetChildIndex)(ADMItemRef inItem, ASInt32 inChildID);

Description
The GetChildIndex() 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,
} ADMSpinEditPopupChildID;
Once a reference to a child item is obtained, any of the ADM Item suite functions can be used to access it.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inChildID</code></td>
<td>ID of child of composite item to return.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>outPluginRef</code></td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td><code>outCursorID</code></td>
<td>Resource ID of the cursor to be displayed when the mouse position is inside the item.</td>
</tr>
<tr>
<td><code>outCursorName</code></td>
<td>Not currently used.</td>
</tr>
</tbody>
</table>

**Returns**

None.
See also

`sADMItem->SetCursorID()`

---

**sADMItem->GetDestroyProc()**

Get the ADM destroy function being used for the item

```c
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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

**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 ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>outPictureResID</code></td>
<td>Resource ID for the picture. If the item does not have a disabled picture, it returns 0.</td>
</tr>
<tr>
<td><code>outPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

**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 ASAPI (*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.
The ADM Item Suite

ADM Item Suite Functions

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

```c
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

Returns

`true` if `inItem` has the rollover property; `false` otherwise.

See also

- `sADMItem->SetHasRolloverProperty()`
- `sADMItem->GetRolloverPicture()`
- `sADMItem->GetRolloverPictureID()`
- `sADMItem->IsInRolloverState()`
- `sADMItem->GetHierarchyList()`

`sADMItem->GetHierarchyList()`

Get the hierarchy list for an item

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

Returns

An ADM hierarchy list.

`sADMItem->GetID()`

Get the ID of an item

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

### Returns

The ID of the resource used to create `inItem`.

---

#### `sADMItem->GetIntValue()` Get the integer value of an item

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

### Returns

The integer value of `inItem`.

### See also

- `sADMItem->SetIntValue()`
- `sADMItem->GetItemStyle()`

---

#### `sADMItem->GetItemStyle()` Get the style of an item

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

### Returns

The item style. Type: `ADMItemStyle` (see `ADMItem.h`)
See also

sADMItem->SetItemStyle()

sADMItem->GetItemType()

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()

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()

float ASAPI (*GetLargeIncrement)(ADMItemRef inItem);

Get the large increment value of an item
**Description**

This function returns the large increment value of the item. It is only used for scroll bar items.

**Parameters**

<table>
<thead>
<tr>
<th><strong>inItem</strong></th>
<th>An ADM item.</th>
</tr>
</thead>
</table>

**Returns**

The large increment value of **inItem**.

**See also**

- `sADMItem->SetLargeIncrement()`
- `sADMItem->GetSmallIncrement()`
- `sADMItem->SetSmallIncrement()`
- `sADMItem->GetList()`

---

**sADMItem->GetList()**

Get the list of an item

```c
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:

```c
#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.

```c
#define kADMUserType "ADM User Type"
```

**Parameters**

<table>
<thead>
<tr>
<th><strong>inItem</strong></th>
<th>An ADM item.</th>
</tr>
</thead>
</table>

**Returns**

**inItem**’s ADM List object.
**sADMItem->GetLocalRect()**

Get the size of an item

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>outLocalRect</td>
<td>The size of the dialog item in ((0,0))-based dimensions. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMItem->SetLocalRect()`

---

**sADMItem->GetMask()**

Get the ADM notification mask for an item

```c
ADMActionMask ASAPI (*GetMask)(ADMItemRef inItem);
```

**Description**

The `GetMask()` function returns the mask used for controlling which events will be received by the tracker.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

**Returns**

The mask. Type: `ADMActionMask` (see `ADMTypes.h`)

**See also**

`sADMItem->SetMask()`

---

**sADMItem->GetMaxFixedValue()**

Get the maximum fixed value of an item

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

Returns

The maximum fixed value of `inItem`. Type: `ASFixed` (see `ASTypes.h`)

See also

`sADMItem->SetMaxFixedValue()`

---

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

Returns

The maximum float value for `inItem`.

See also

`sADMItem->SetMaxFloatValue()`

---

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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, it is converted to a floating point decimal value.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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 an integer.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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.
The ADM Item Suite

ADM Item Suite Functions

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 ASAPI (*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 ASAPI (*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.
### ADM Item Suite Functions

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

#### 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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>outPictureResID</code></td>
<td>Resource ID for the picture. If the item does not have a disabled picture, it returns 0.</td>
</tr>
<tr>
<td><code>outPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

#### Returns

None.

#### See also

- `sADMItem->SetPictureID()`  
- `sADMItem->GetDisabledPicture()`  
- `sADMItem->GetDisabledPictureID()`  
- `sADMItem->GetPictureID()`  
- `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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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

gADMItem->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

gADMItem->SetRolloverPicture()
gADMItem->GetRolloverPictureID()
gADMItem->GetHasRolloverProperty()
gADMItem->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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>outPictureResID</td>
<td>Resource ID for the picture.</td>
</tr>
<tr>
<td>outPictureResName</td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- `sADMItem->SetRolloverPictureID()`
- `sADMItem->GetRolloverPicture()`
- `sADMItem->GetHasRolloverProperty()`
- `sADMItem->IsInRolloverState()`
- `sADMItem->GetSelectedPicture()`

### sADMItem->GetSelectedPicture()

Gets the picture currently selected in the item

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

Returns

An ADM icon.

See also

- `sADMItem->SetSelectedPicture()`
- `sADMItem->GetPicture()`
- `sADMItem->GetPictureID()`
- `sADMItem->GetSelectedPictureID()`
sADMItem->GetSelectedPictureID()

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

<table>
<thead>
<tr>
<th>inItem</th>
<th>An ADM item.</th>
</tr>
</thead>
<tbody>
<tr>
<td>outPictureResID</td>
<td>Resource ID for the picture. If the item does not have a picture, it returns 0.</td>
</tr>
<tr>
<td>outPictureResName</td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMItem->SetSelectedPictureID()
sADMItem->GetPicture()
sADMItem->GetPictureID()
sADMItem->GetSelectedPicture()
sADMItem->GetDisabledPicture()
sADMItem->GetDisabledPictureID()

sADMItem->GetSelectionRange()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>outSelStart</code></td>
<td>Beginning of selection.</td>
</tr>
<tr>
<td><strong>NOTE:</strong></td>
<td><code>outSelStart</code> precedes the first selected character and thus might be 0.</td>
</tr>
<tr>
<td><code>outSelEnd</code></td>
<td>End of selection. Equal to the last selected character and can be no more than the item's text length.</td>
</tr>
</tbody>
</table>

### 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

## Returns

The small increment value of `inItem`.

## See also

- `sADMItem->SetSmallIncrement()`
- `sADMItem->GetLargeIncrement()`
- `sADMItem->SetLargeIncrement()`

### `sADMItem->GetText()`

Get the item’s text

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>outText</code></td>
<td><code>inItem</code>'s text.</td>
</tr>
<tr>
<td><code>inMaxLength</code></td>
<td>Size of the already allocated buffer for <code>outText</code>.</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

- `sADMItem->SetText()`

### `sADMItem->GetTextLength()`

Get the length of an item's text

```c
ASInt32 ASAPI (*GetTextLength)(ADMItemRef inItem);
```

### Description

The `GetTextLength()` function gets the number of characters in `inItem`'s text.
## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

## 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.

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

### 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.

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>
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 ASAPI (*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 ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inIgnoreForceRoman</code></td>
<td>If <code>true</code>, Roman script forcing of fonts is overridden; if <code>false</code>, it is not.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMItem->Create()`
`sADMDrawer->Create()`

`sADMItem->Invalidate()`  Invalidate the area of an item

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>
The ADM Item Suite
ADM Item Suite Functions

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inInvalRect</td>
<td>A rectangle within inItem Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>
**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. |
The ADM Item Suite

ADM Item Suite Functions

Returns

- **true** if `inItem` is visible; **false** otherwise.

See also

- `sADMItem->Show()`

---

### sADMItem->Known()

**Make an item known**

```c
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**

```c
void ASAPI (*LocalToScreenPoint)(ADMItemRef inItem, ASPoint* ioPoint);
```

**Description**

The `LocalToScreenPoint()` function converts a point in `inItem` to a point in screen coordinates.
Parameters

<table>
<thead>
<tr>
<th>inItem</th>
<th>An ADM item.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioPoint</td>
<td>The point. Type: ASPoint (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMItem->ScreenToLocalPoint()
sADMDialog->LocalToScreenPoint()

---

**sADMItem->LocalToScreenRect()**

Convert an item rectangle to screen coordinates

```c
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

<table>
<thead>
<tr>
<th>inItem</th>
<th>An ADM item.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ioRect</td>
<td>The rect. Type: ASRect (see ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMItem->ScreenToLocalRect()
sADMDialog->LocalToScreenRect()

---

**sADMItem->Move()**

Set the absolute position of an item

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inHorizPosition</code></td>
<td>New horizontal screen coordinate for <code>inItem</code>.</td>
</tr>
<tr>
<td><code>inVertPosition</code></td>
<td>New vertical screen coordinate for <code>inItem</code>.</td>
</tr>
</tbody>
</table>

**Returns**

None.

---

**sADMItem->RemoveItem()**

Remove an item from a group

```c
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 disable 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 `kADMMItemGroupType`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inGroup</code></td>
<td>An ADM item group.</td>
</tr>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>ioPoint</td>
<td>The point. Type: <code>ASPoint</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMItem->LocalToScreenPoint()`  
- `sADMItem->ScreenToLocalRect()`  
- `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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>ioRect</td>
<td>The rect. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMItem->LocalToScreenRect()`  
- `sADMItem->ScreenToLocalRect()`  
- `sADMDialog->ScreenToLocalRect()`
The ADM Item Suite
ADM Item Suite Functions

sADMItem->SelectAll()

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()

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

<table>
<thead>
<tr>
<th>inItem</th>
<th>An ADM item.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inNotifierType</td>
<td>The type of notification to send. (See ADMNotifier.h for the notification constants.)</td>
</tr>
</tbody>
</table>

Returns
None.
**sADMItem->SetAllowMath()**

Sets whether an item uses math in computing entered values

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inAllow</code></td>
<td><code>true</code> if math can be used in computing user input values for an item; <code>false</code> otherwise.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMItem->GetAllowMath()`

**sADMItem->SetAllowUnits()**

Sets whether an item uses units in numeric values when value is entered by user

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inAllowUnits</code></td>
<td><code>true</code> if units are used for an item when the value is entered by the user; <code>false</code> otherwise.</td>
</tr>
</tbody>
</table>

**Returns**

None.
See also

`sADMItem->GetAllowUnits()`

---

`sADMItem->SetBackColor()`

Sets an item's background color

```c
void ASAPI (*SetBackColor)(ADMItemRef inItem, ADMColor inColor);
```

**Description**

The *SetBackColor()* function sets an item's background color.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inColor</code></td>
<td><code>inItem</code>'s background color. Type: <em>ADMColor</em> (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

See also

`sADMItem->GetBackColor()`

---

`sADMItem->SetBooleanValue()`

Set the boolean value of an item

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inValue</code></td>
<td><code>true</code> or <code>false</code> to indicate the state of the item.</td>
</tr>
</tbody>
</table>

**Returns**

None.

See also

`sADMItem->GetBooleanValue()`
sADMItem->SetBoundsRect() Set the absolute position and size of an item

```c
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:

```c
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.

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inBoundsRect</code></td>
<td>The size and position of an item in its containing dialog's coordinate space. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMItem->GetBoundsRect()`
- `sADMItem->Move()`

sADMItem->SetCursorID() Set the item’s cursor ID

```c
ASBoolean ASAPI (*SetCursorID)(ADMItemRef inItem, SPPluginRef inPluginRef, ASInt32 inCursorID, const char* inCursorName);
```
The ADM Item Suite

ADM Item Suite Functions

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inPluginRef</td>
<td>Plug-in reference.</td>
</tr>
<tr>
<td>inCursorID</td>
<td>ID of a platform native cursor resource.</td>
</tr>
<tr>
<td>inCursorName</td>
<td>Cursor name. Can also be <strong>NULL</strong>.</td>
</tr>
</tbody>
</table>

**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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inDestroyProc</td>
<td>A callback with the following signature (see ADMItem.h):</td>
</tr>
<tr>
<td></td>
<td>ADMItemDestroyProc(ADMItemRef inItem);</td>
</tr>
</tbody>
</table>
The ADM Item Suite

ADM Item Suite Functions

Returns
None.

See also
sADMItem->GetDestroyProc()

sADMItem->SetDisabledPicture()
Set the item’s disabled picture

void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inPicture</td>
<td>ADM icon that is to be displayed when inItem is disabled.</td>
</tr>
</tbody>
</table>

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 ASAPI (*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.
The ADM Item Suite

ADM Item Suite Functions

Not all ADM items have pictures. ADM items for which this function sets a picture resource are:

```c
#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:

```c
#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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inPictureResID</code></td>
<td>Resource ID for the picture.</td>
</tr>
<tr>
<td><code>inPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

**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.

```c
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.
The ADM Item Suite

Parameters

\[ \text{inItem} \quad \text{An ADM item.} \]

\[ \text{inDrawProc} \quad \text{A callback with the following signature (see \text{ADMItem.h}):} \]

\[ \text{ADMItemDrawProc} \left( \text{ADMItemRef inItem, ADMDrawerRef inDrawer} \right) ; \]

* \text{ADMDrawerRef} \text{ argument is passed to ADM Drawer suite functions to indicated where the imaging is to occur.}*

\[ \text{Returns} \]

None.

See also

\[ \text{sADMItem->GetDrawProc()} \]

\[ \text{sADMItem->GetFixedValue()} \]

\[ \text{sADMItem->SetFloatToTextProc()} \]

\[ \text{Set the fixed value of an item} \]

void ASAPI (*SetFixedValue)(ADMItemRef inItem, ASFixed inValue);

Description

The \text{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

\[ \text{inItem} \quad \text{An ADM item.} \]

\[ \text{inValue} \quad \text{A fixed value for inItem. Type: ASFixed (see \text{ASTypes.h})} \]

Returns

None.

See also

\[ \text{sADMItem->GetFloatToTextProc()} \]

\[ \text{sADMItem->GetFixedValue()} \]

\[ \text{sADMItem->SetFloatToTextProc()} \]

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 ADMItem.h):
  
  ```c
  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

```c
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**

```c
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 `ADMTypes.h`)

**Returns**

None.

**See also**

- `sADMItem->GetFont()`

### `sADMItem->SetForeColor()`

**Set the item's foreground color**

```c
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 `ADMTypes.h`)

**Returns**

None.

**See also**

- `sADMItem->GetForeColor()`
sADMItem->SetHasRolloverProperty()

Set whether the item has the rollover property

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inRollover</td>
<td>If <code>true</code>, <code>inItem</code> has the rollover property; if <code>false</code>, it doesn’t.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMItem->GetHasRolloverProperty()`
- `sADMItem->SetRolloverPicture()`
- `sADMItem->SetRolloverPictureID()`
- `sADMItem->SetInRolloverState()`

---

sADMItem->SetInRolloverState()

Set whether the item is in the rollover state

```c
void ASAPI (*SetInRollOverState)(ADMItemRef inItem, ASBoolean inRolloverState);
```

**Description**

The `SetInRollOverState()` sets whether `inItem` is currently rolled over.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inRolloverState</td>
<td>If <code>true</code>, <code>inItem</code> is in the rollover state; if <code>false</code>, it isn’t.</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>inItem</th>
<th>An ADM item.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inValue</td>
<td>The integer value for inItem</td>
</tr>
</tbody>
</table>

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 ADMItem.h)

Returns

None.

See also

sADMItem->GetItemStyle()  
sADMItem->Update()  
sADMItem->SetItemType()

**sADMItem->SetItemType()**
Set the type of an item

```c
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:

```c
#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.
Returns

None.

See also

sADMItem->ItemType()
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

<table>
<thead>
<tr>
<th>inItem</th>
<th>An ADM item.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inJustify</td>
<td>Text justification for inItem. Type: ADMJustify (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMItem->GetJustify()
sADMItem->SetLargeIncrement() Set the large increment value of an item

```c
void ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inIncrement</code></td>
<td>The large increment value for <code>inItem</code></td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMItem->GetLargeIncrement()`
- `sADMItem->GetSmallIncrement()`
- `sADMItem->SetSmallIncrement()`
- `sADMItem->SetLocalRect()`

sADMItem->SetLocalRect() Set the size of an item

```c
void ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inLocalRect</code></td>
<td>The size for the item in ((0,0))-based dimensions. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.
See also

sADMItem->GetLocalRect()

---

sADMItem->SetMask()

Set the ADM notification mask for an item

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inActionMask</code></td>
<td>The mask. Type: <code>ADMActionMask</code> (see <code>ADMTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

See also

sADMItem->GetMask()

---

sADMItem->SetMaxFixedValue()

Set the maximum fixed value of an item

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inValue</code></td>
<td>The maximum fixed value for <code>inItem</code>. Type: <code>ASFixed</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inValue</td>
<td>The maximum float value for inItem</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inValue</td>
<td>The maximum integer value of inItem</td>
</tr>
</tbody>
</table>

Returns
None.
The ADM Item Suite
ADM Item Suite Functions

See also
sADMItem->GetMaxIntValue()

sADMItem->SetMaxTextLength()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inLength</td>
<td>Maximum number of characters that inItem’s text property can have.</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMItem->GetMaxTextLength()

sADMItem->SetMinFixedValue()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inValue</td>
<td>The lower threshold on the value of inItem. Type: ASFixed (see ASTypes.h)</td>
</tr>
</tbody>
</table>
Returns

None.

See also

sADMItem->GetMinFixedValue()

---

sADMItem->SetMinFloatValue()

**Set the minimum float value of an item**

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inValue</code></td>
<td>Lower threshold on the value of <code>inItem</code></td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMItem->GetMinFloatValue()

---

sADMItem->SetMinIntValue()

**Set the minimum integer value of an item**

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inValue</code></td>
<td>Minimum integer value of <code>inItem</code></td>
</tr>
</tbody>
</table>
The ADM Item Suite

ADM Item Suite Functions

Returns
None

See also
sADMItem->GetMinIntValue()

---

`sADMItem->SetNotifierData()`

Set notifier data of an item

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inData</code></td>
<td>Custom notification data. Type: <code>ADMUserData</code> (see <code>ADMTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMItem->GetNotifierData()

---

`sADMItem->SetNotifyProc()`

Set the ADM notification function to use for the item

```c
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.
The ADM Item Suite

ADM Item Suite Functions

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inItem</strong></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><strong>inNotifyProc</strong></td>
<td>Callback with the following signature (see ADMItem.h):</td>
</tr>
<tr>
<td></td>
<td>ADMItemNotifyProc(inItem, inNotifier);</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMItem->GetNotifyProc()`

---

**sADMItem->SetPicture()**

Sets a picture based on an icon

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inItem</strong></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><strong>inPicture</strong></td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>

**Returns**

None

**See also**

- `sADMItem->GetPicture()`
- `sADMItem->SetSelectedPictureID()`
- `sADMItem->SetPictureID()`
- `sADMItem->SetSelectedPicture()`
sADMItem->SetPictureID()

Set the item’s picture ID

```
void ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inPictureResID</code></td>
<td>Resource ID for the picture.</td>
</tr>
<tr>
<td><code>inPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

```
sADMItem->GetPictureID()
sADMItem->SetPicture()
sADMItem->SetSelectedPictureID()
sADMItem->SetSelectedPicture()
sADMItem->SetDisabledPicture()
sADMItem->SetDisabledPictureID()
sADMItem->SetPluginRef()
```

sADMItem->SetPluginRef()

Set the plug-in that created an item

```
void ASAPI (*SetPluginRef)(ADMItemRef inItem, SPPluginRef
inPluginRef);
```
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inPluginRef</code></td>
<td>Plug-in reference.</td>
</tr>
</tbody>
</table>

### Returns

None.

### See also

`sADMItem->GetPluginRef()`

---

**sADMItem->SetPopupDialog()**

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inPopupItemID</code></td>
<td>Pop-up dialog ID. (See <code>ADMResource.h</code>).</td>
</tr>
<tr>
<td><code>inDialog</code></td>
<td>An ADM dialog.</td>
</tr>
</tbody>
</table>

### Returns

None.
See also

sADMItem->GetPopupDialog()

sADMItem->SetPrecision()

Set the precision of an item

```c
void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inNumberOfDecimalPlaces</code></td>
<td>Number of digits that the user is allowed to enter following a decimal point.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMItem->GetPrecision()

sADMItem->SetRolloverPicture()

Set the rollover picture for an item

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inPicture</td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- sADMItem->GetRolloverPicture()
- sADMItem->SetRolloverPictureID()
- sADMItem->SetHasRolloverProperty()
- sADMItem->SetInRolloverState()

sADMItem->SetRolloverPictureID()

Set the rollover picture ID of an item

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inPictureResID</td>
<td>Resource ID for the picture.</td>
</tr>
<tr>
<td>inPictureResName</td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- sADMItem->GetRolloverPictureID()
- sADMItem->SetRolloverPicture()
- sADMItem->SetHasRolloverProperty()
- sADMItem->SetInRolloverState()
sADMItem->SetSelectedPicture()

Set the item's selected picture

```c
void ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inPicture</code></td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>

**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

```c
void ASAPI (*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:

```c
#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:

```c
#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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inPictureResID</code></td>
<td>Resource ID for the picture. If the item does not have a picture, it returns 0.</td>
</tr>
<tr>
<td><code>inPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

**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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inSelStart</code></td>
<td>Beginning of selection.</td>
</tr>
<tr>
<td><code>inSelEnd</code></td>
<td>End of selection. Equal to the last selected character and can be no more than the item's text length.</td>
</tr>
</tbody>
</table>

**NOTE:** `inSelStart` precedes the first selected character and thus might be 0.
The ADM Item Suite

ADM Item Suite Functions

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

<table>
<thead>
<tr>
<th>Parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
</tr>
<tr>
<td>inIncrement</td>
</tr>
</tbody>
</table>

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>
<thead>
<tr>
<th>Item Type</th>
<th>Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>kADMFrameType</td>
<td>Text is the frame title.</td>
</tr>
<tr>
<td>Item Type</td>
<td>Initialization</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
</tr>
<tr>
<td>kADMTextEditPopupType</td>
<td>The text is the editable text.</td>
</tr>
<tr>
<td>kADMSpinEditPopupType</td>
<td></td>
</tr>
<tr>
<td>kADMSpinEditScrollingPopupType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupSpinEditControlType</td>
<td></td>
</tr>
<tr>
<td>kADMSpinEditType</td>
<td></td>
</tr>
<tr>
<td>kADMTextEditType</td>
<td></td>
</tr>
<tr>
<td>kADMTextEditMultiLineType</td>
<td></td>
</tr>
<tr>
<td>kADMTextEditScrollingPopupType</td>
<td></td>
</tr>
<tr>
<td>kADMTextEditCheckBoxType</td>
<td>The text is displayed to the right of the check box.</td>
</tr>
<tr>
<td>kADMTextEditPushButtonType</td>
<td>The text is displayed within the push button.</td>
</tr>
<tr>
<td>kADMTextEditRadioButtonType</td>
<td>The text is displayed to the right of the radio button.</td>
</tr>
<tr>
<td>kADMTextEditStaticType</td>
<td>The text is displayed.</td>
</tr>
<tr>
<td>kADMTextEditStaticMultiLineType</td>
<td></td>
</tr>
<tr>
<td>kADMUserType</td>
<td>Unknown.</td>
</tr>
<tr>
<td>kADMUserType</td>
<td>The text is unused.</td>
</tr>
<tr>
<td>kADMUserType</td>
<td></td>
</tr>
<tr>
<td>kADMPicturePushButtonType</td>
<td></td>
</tr>
<tr>
<td>kADMPictureRadioButtonType</td>
<td></td>
</tr>
<tr>
<td>kADMPictureStaticType</td>
<td></td>
</tr>
<tr>
<td>kADMPicturePushButtonType</td>
<td></td>
</tr>
<tr>
<td>kADMPictureCheckBoxType</td>
<td></td>
</tr>
<tr>
<td>kADMListBoxType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupListType</td>
<td></td>
</tr>
<tr>
<td>kADMHierarchyListBoxType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupMenuType</td>
<td></td>
</tr>
<tr>
<td>kADMScrollingPopupListType</td>
<td></td>
</tr>
<tr>
<td>kADMResizeType</td>
<td></td>
</tr>
<tr>
<td>kADMSliderType</td>
<td></td>
</tr>
<tr>
<td>kADMScrollbarType</td>
<td></td>
</tr>
<tr>
<td>kADMScrollBarType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupControlType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupControlButtonType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupControlType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupControlButtonType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupControlType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupControlButtonType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupControlType</td>
<td></td>
</tr>
<tr>
<td>kADMPopupControlButtonType</td>
<td></td>
</tr>
<tr>
<td>kADMProgressbar</td>
<td></td>
</tr>
<tr>
<td>kAADMCrossingArrows</td>
<td></td>
</tr>
</tbody>
</table>
The ADM Item Suite

ADM Item Suite Functions

sADMItem->SetTextToFloatProc()

Sets an item's procedure for converting a text string to a floating point value.

```c
void ASAPI (*SetTextToFloatProc)(ADMItemRef inItem,
                                 ADMItemTextToFloatProc inProc);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inText</td>
<td>inItem’s text.</td>
</tr>
</tbody>
</table>

**Description**

The `SetTextToFloatProc()` function sets the text to floating point conversion procedure for `inItem`. See FloatToText and TextToFloat Functions for more information.

**Returns**

None.

**See also**

- `sADMItem->GetText()`
- `sADMItem->GetTextToFloatProc()`
- `sADMItem->DefaultTextToFloat()`

sADMItem->SetTrackProc()

Set the ADM tracker function to use for the item.

```c
void ASAPI (*SetTrackProc)(ADMItemRef inItem,
                           ADMItemTrackProc inTrackProc);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inProc</td>
<td>A callback with the following signature (see ADMItem.h):</td>
</tr>
<tr>
<td></td>
<td>ASBoolean ADMItemTextToFloatProc(ADMItemRef inItem, const char* inText, float* outValue);</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMItem->GetTextToFloatProc()`
- `sADMItem->DefaultTextToFloat()`
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
</tbody>
</table>
| `inTrackProc` | A callback with the following signature (see ADMItem.h):
  
  ```c
  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 except 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

```c
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: ADMUnits (see ADMTypes.h)

Returns

None.

See also

- `sADMItem->GetUnits()`

---

`sADMItem->SetUserData()` Set the user data pointer for an item

```c
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()

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()

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inShow</td>
<td>If true, selected units for the field are show; if false, they are not.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inWidth</td>
<td>Width of inItem (in pixels).</td>
</tr>
<tr>
<td>inHeight</td>
<td>Height of inItem (in pixels).</td>
</tr>
</tbody>
</table>

Returns

None.
**sADMItem->Update()**

Force an update of an item

```c
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

```c
ASBoolean ASAPI (*WasPercentageChange)(ADMItemRef inItem);
```

**Description**

The `WasPercentageChange()` function 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 `lin+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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inItem</code></td>
<td>An ADM item.</td>
</tr>
<tr>
<td><code>inEnable</code></td>
<td>If <strong>true</strong>, tool tip is enabled; if <strong>false</strong>, tool tip is disabled.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMItem->IsTipEnabled()`

### sADMItem->GetHelpID()

Get the Help ID of an item

```c
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

```c
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

```c
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

```c
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

```c
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()

void ASAPI (*SetTipString)(ADMItemRef inItem, const char* inTipString);

Description
The SetTipString() function sets the tool tip of inItem to the supplied C string.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inTipString</td>
<td>Tool tip of inItem</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMItem->GetTipString()
sADMItem->GetTipStringLength()

sADMItem->ShowToolTip()

void ASAPI (*ShowToolTip)(ADMItemRef inItem, const APoint* inWhere);

Description
The ShowToolTip() function displays the tool tip for inItem as if the mouse were held over it.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inItem</td>
<td>An ADM item.</td>
</tr>
<tr>
<td>inWhere</td>
<td>Position for display of the tool tip. Type: APoint (See ASTypes.h)</td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMItem->HideToolTip()
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:

```c
#define kADMListSuite "ADM List Suite"
```

with the version constant:

```c
#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:

```c
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()`:

```cpp
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:

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

You can also create each entry with the `sADMLList->InsertEntry()` function followed by the `sADMEntry->SetText()` and `sADMEntry->SetID()` functions:

```cpp
for (index = 0; index < kNumberEntries; index++) {
    char menuText[255];
    ADMEntryRef entry = sADMLList->InsertEntry(theItemList, index);
    sBasic->GetIndexString(thePlugin, 16000, index, menuText, 255);
    sADMEntry->SetText(entry, menuText);
    sADMEntry->SetID(entry, index);
}
```

**NOTE:** List indices are 0-based.

To get the currently selected item of a single selection list, use the `sADMLList->GetActiveEntry()` function and then get the entry’s index:

```cpp
int GetListValue(ADMItem theListItem) {
    ADMList theList = sADMItem->GetList(theListItem);
    ADMEntry theEntry = sADMLList->GetActiveEntry(theItemList);
    return sADMEntry->GetID(theEntry);
}
```

To get each selected item in a multiple selection list, get the selection count and iterate through the selections:

```cpp
int count = NumberOfSelectedEntries(theList);
for (index = 0; index < count; index++) {
    ADMEntryRef entry = sADMLList->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

```c
ADMEntryRef 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**

<table>
<thead>
<tr>
<th><code>inList</code></th>
<th>An ADM list.</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inText</code></td>
<td>Entry text to search for.</td>
</tr>
</tbody>
</table>

**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

```c
ADMEntryRef 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**

<table>
<thead>
<tr>
<th>Table 15.2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

**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

```c
ADMEntryDestroyProc ASAPI (*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**

<table>
<thead>
<tr>
<th>Table 15.3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

**Returns**

The Destroy proc for `inList`. If you have not called `sADMList->SetDestroyProc()`, returns `NULL` (not the default Destroy proc). Type: `ADMEntryDestroyProc` (see `ADMList.h`)

**See also**

`sADMList->SetDestroyProc()`

---

### `sADMList->GetDrawProc()`

Get the ADM drawing function being used for the list’s entries

```c
ADMEntryDrawProc 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 `sADMEntry->DefaultDraw()`.

Parameters

**Table 15.4**

| `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: `ADMEntryDrawProc` (see `ADMList.h`)

See also

- `sADMEntry->DefaultDraw()`
- `sADMList->SetDrawProc()`

`sADMList->GetEntry()`

Get an entry by ID

`ADMEntryRef 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 `sADMEntry->GetID()` function. If you keep this ID, you can pass it to this function at a later time and retrieve the item.

Parameters

**Table 15.5**

| `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

- `sADMEntry->GetID()`
The ADM List Suite

ADM List Suite Functions

**sADMList->GetEntryHeight()**

Get the height of a list’s entries

```c
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**

<table>
<thead>
<tr>
<th>TABLE 15.6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

**Returns**

Height of `inList`’s entries.

**See also**

`sADMList->SetEntryHeight()`

**sADMList->GetEntryTextRect()**

Get the edit text rectangle for a list

```c
void ASAPI (*GetEntryTextRect)(ADMListRef inList, ASRect* outRect);
```

The `GetEntryTextRect()` function gets the location of the edit-in-place text item of `inList`.

**Parameters**

<table>
<thead>
<tr>
<th>TABLE 15.7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
<tr>
<td><strong>outRect</strong></td>
</tr>
</tbody>
</table>

**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

**Table 15.8**

| 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

ADMEntryInitProc 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

**Table 15.9**

| 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: `ADMEntryInitProc` (see `ADMList.h`)

See also

sADMList->SetInitProc()
sADMList->GetItem()

Get the item reference for a list

```c
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**

`TABLE 15.10`

<table>
<thead>
<tr>
<th>inList</th>
<th>An ADM list.</th>
</tr>
</thead>
</table>

**Returns**

The ADM item to which `inList` belongs.

---

sADMList->GetMask()

Get the filter on events received by the entries’ tracker

```c
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**

`TABLE 15.11`

<table>
<thead>
<tr>
<th>inList</th>
<th>An ADM list.</th>
</tr>
</thead>
</table>

**Returns**

The mask. Type: `ADMActionMask` (see `ADMTypes.h`)

**See also**

`sADMList->SetMask()`

---

sADMList->GetMenuID()

Get the menu resource ID of a list

```c
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

<table>
<thead>
<tr>
<th>TABLE 15.12</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

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

```c
ADMUserData ASAPI (*GetNotifierData)(ADMListRef inList);
```

Description

The `GetNotifierData()` function returns the notification data of `inList`.

Parameters

<table>
<thead>
<tr>
<th>TABLE 15.13</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

Returns

Custom notification data. Type: `ADMUserData` (see ADMTypes.h)

See also

`sADMList->SetNotifierData()`

---

`sADMList->GetNotifyProc()` Get the ADM notification function being used for entries

```c
ADMEntryNotifyProc 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 `sADMEntry->DefaultNotify()`.

**Parameters**

<table>
<thead>
<tr>
<th>Table 15.14</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

**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: `ADMEntryNotifyProc` (see `ADMList.h`)

**See also**

- `sADMList->SetNotifyProc()`
- `sADMEntry->DefaultNotify()`

### `sADMList->GetTrackProc()`

Get the ADM tracker function being used for the list's entries

```c
ADMDiallogTrackProc ASAPI (*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 `sADMEntry->DefaultTrack()` function.

**Parameters**

<table>
<thead>
<tr>
<th>Table 15.15</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

**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: `ADMDiallogTrackProc` (see `ADMList.h`)

**See also**

- `sADMList->SetTrackProc()`
sADMLList->GetUserData()  
Get the user data value for a list

ADMUserData ASAPI (*GetUserData)(ADMLListRef 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

<table>
<thead>
<tr>
<th>TABLE 15.16</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

Returns
A 4-byte value that ADM keeps with inList. Type: ADMUserData (see ADMTypes.h)

See also
sADMLList->SetUserData()

sADMLList->IndexEntry()  
Get an entry by index

ADMEntryRef ASAPI (*IndexEntry)(ADMLListRef 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 sADMLList->NumberOfEntries(), you can iterate through all of a
list's entries (see example).

Parameters

<table>
<thead>
<tr>
<th>TABLE 15.17</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
<tr>
<td><strong>inIndex</strong></td>
</tr>
</tbody>
</table>

Returns
The ADM entry at inIndex.
See also

`sADMList->NumberOfEntries()`

Example

```c
ADMListRef list;
ADMEntryRef 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

```c
ADMEntryRef 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

**TABLE 15.18**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM list.</td>
</tr>
<tr>
<td><code>inSelectionIndex</code></td>
<td>The 0-based index of the selected entry in <code>inList</code> for which a reference is requested.</td>
</tr>
</tbody>
</table>

Returns

A selected entry.

See also

`sADMList->NumberOfSelectedEntries()`
**Example**

```c
ADMListRef list;
ADMEntryRef 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

```c
ADMEntryRef ASAPI (*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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM list.</td>
</tr>
<tr>
<td><code>inIndex</code></td>
<td>The index where to insert the new entry into <code>inList</code>.</td>
</tr>
</tbody>
</table>

**Returns**

A reference to the new entry.

**See also**

- `sADMList->SetInitProc()`
- `sADMList->RemoveEntry()`

**sADMList->NumberOfEntries()**

Get the number of entries in a list

```c
ASInt32 ASAPI (*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**

<table>
<thead>
<tr>
<th>Table 15.20</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

**Returns**

The number of entries in `inList`.

**See also**

`sADMList->IndexEntry()`

---

### sADMList->NumberOfSelectedEntries()

Get the number of selected entries in a list

```cpp
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**

<table>
<thead>
<tr>
<th>Table 15.21</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>

**Returns**

Number of selected entries in `inList`.

**See also**

`sADMList->IndexSelectedEntry()`

---

### sADMList->PickEntry()

Select an entry at a specified point

```cpp
ADMEntryRef 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

**TABLE 15.22**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM list.</td>
</tr>
<tr>
<td><code>inPoint</code></td>
<td>A point in <code>inList</code>. Type: ASPoint (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns

The ADM entry at `inPoint` in `inList`.

`sADMList->RemoveEntry()` Remove an entry from a list

```c
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

**TABLE 15.23**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM list.</td>
</tr>
<tr>
<td><code>inIndex</code></td>
<td>Index of the entry to remove.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMList->InsertEntry()`
`sADMList->SetDestroyProc()`

`sADMList->SelectByText()` Searches for and selects text

```c
void ASAPI (*SelectByText)(ADMListRef inList, const char* inFindText);
```

Description

The `SelectByText()` function searches for and selects text.
Parameters

Table 15.24

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM list.</td>
</tr>
<tr>
<td>inFindText</td>
<td>Text to search for and select.</td>
</tr>
</tbody>
</table>

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

Table 15.25

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM list.</td>
</tr>
<tr>
<td>inColor</td>
<td>The color to which to set the background of inList. Type: ADMColor (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

sADMList->SetDestroyProc()

Set the ADM destroy function to use for the list’s entries

void ASAPI (*SetDestroyProc)(ADMListRef inList, ADMEntryDestroyProc 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

<table>
<thead>
<tr>
<th>Table 15.26</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
<tr>
<td><strong>inDestoryProc</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMList->GetDestroyProc()

sADMList->SetDrawProc() Set the ADM drawing function to use for the list’s entries

```c
void ASAPI (*SetDrawProc)(ADMListRef inList, ADMEntryDrawProc 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 indicated where imaging occurs.

To call the default drawing function for an entry, use `sADMEntry->DefaultDraw()`.

See Using Event Callbacks in Chapter 1, “ADM Overview” for more information.

Parameters

<table>
<thead>
<tr>
<th>Table 15.27</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
<tr>
<td><strong>inDrawProc</strong></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Table 15.28</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
<tr>
<td><strong>inHeight</strong></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Table 15.29</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
<tr>
<td><strong>inRect</strong></td>
</tr>
</tbody>
</table>

Returns
None.

See also
sADMList->GetEntryTextRect()
**sADMList->SetEntryWidth()**

Set the width of a column of entries

```c
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**

<table>
<thead>
<tr>
<th>Table 15.30</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
<td>An ADM list.</td>
</tr>
<tr>
<td><strong>inWidth</strong></td>
<td>Width of a column in <code>inList</code> if the list is part of a listbox item and has the style <code>kADMTileListBoxStyle</code>. For any other list type or style, sets the width of the entries' local and bounds rect.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMList->GetEntryWidth()`

---

**sADMList->SetInitProc()**

Set the ADM init function to use for entries

```c
void ASAPI (*SetInitProc)(ADMListRef inList, ADMEntryInitProc 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.
The ADM List Suite

ADM List Suite Functions

---

**Parameters**

**Table 15.31**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM list.</td>
</tr>
</tbody>
</table>
| `inInitProc` | A callback with the following signature (see ADMItem.h):
|             | `ADMEntryInitProc(ADMEntryRef inEntry);` |

**Returns**

None.

**See also**

`sADMList->GetInitProc()`

### sADMList->SetMask()

Set a filter on events received by the entries' tracker

```c
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**

**Table 15.32**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM list.</td>
</tr>
<tr>
<td><code>inActionMask</code></td>
<td>The mask. Type: <code>ADMActionMask</code> (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMList->GetMask()`

### sADMList->SetMenuID()

Set the menu resource ID of a list

```c
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

<table>
<thead>
<tr>
<th>Table 15.33</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
<tr>
<td><strong>inMenuResPlugin</strong></td>
</tr>
<tr>
<td><strong>inMenuResID</strong></td>
</tr>
<tr>
<td><strong>inMenuResName</strong></td>
</tr>
</tbody>
</table>

Returns

None.

See also

* sADMList->GetMenuID()  
* sADMHierarchyList->GetMenuID()

`sADMList->SetNotifierData()`  
Set notifier data for a list entry

```c
void ASAPI (*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

<table>
<thead>
<tr>
<th>Table 15.34</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
<tr>
<td><strong>inUserData</strong></td>
</tr>
</tbody>
</table>

Returns

None.
The ADM List Suite

ADM List Suite Functions

See also

sADMList->GetNotifierData()

sADMList->SetNotifyProc()

Set the ADM notification function to use for the list’s entries

void ASAPI (*SetNotifyProc)(ADMListRef inList, ADMEntryNotifyProc 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

<table>
<thead>
<tr>
<th>Table 15.35</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
</tbody>
</table>
| **inNotifyProc** | A callback with the following signature (see ADMItem.h):

```
ADMEntryNotifyProc(ADMEntryRef inEntry, ADMNotifierRef inNotifier);
```

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. |

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, ADMEntryTrackProc 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inList</code></td>
<td>An ADM list.</td>
</tr>
<tr>
<td><code>inTrackProc</code></td>
<td>A callback with the following signature (see ADMItem.h):&lt;br&gt;<code>ADMEntryTrackProc(ADMEntryRef inEntry, ADMTrackerRef inTracker);</code>&lt;br&gt;Within your track function you can use the ADM Tracker suite functions to access event information. The <code>inTracker</code> argument is passed to the ADM Tracker suite functions to indicate the event for which information is being requested. Your function should return <code>true</code> if ADM should call the entry’s notifier function or <code>false</code> if it shouldn’t.</td>
</tr>
</tbody>
</table>

Returns

None

See also

`sADMList->GetTrackProc()`

`sADMList->SetUserData()` Set the user data value for a list

```c
void ASAPI (*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.
The ADM List Suite

ADM List Suite Functions

Parameters

<table>
<thead>
<tr>
<th>Table 15.37</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inList</strong></td>
</tr>
<tr>
<td><strong>inUserData</strong></td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMList->GetUserData()
### 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](image)

### Accessing the Suite

The ADM List Entry suite is referred to as:

```c
#define kADMListEntrySuite "ADM List Entry Suite"
```
with the version constant:

```c
#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:

```c
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

```c
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

<table>
<thead>
<tr>
<th>inEntry</th>
<th>An ADM list entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inTimer</td>
<td>An ADM timer.</td>
</tr>
</tbody>
</table>

Returns

`sADMListEntry->CreateTimer()`

`sADMListEntry->Activate()`

Make an ADM list entry active or inactive

```c
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

<table>
<thead>
<tr>
<th>inEntry</th>
<th>An ADM list entry.</th>
</tr>
</thead>
<tbody>
<tr>
<td>inActivate</td>
<td>If <code>true</code>, the entry is activated; if <code>false</code>, it is inactivated.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

`sADMListEntry->IsActive()`

`sADMListEntry->AreChildrenSelected()`

Determine whether the children of an ADM list entry are selected

```c
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

Return `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

```c
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

```c
ADMListEntryRef ASAPI (*Create)(ADMHierarchyListRef inList);
```
The ADM List Entry Suite

ADM List Entry Suite Functions

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inList</td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

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 it.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

Returns

The new child list.

See also

sADMListEntry->DeleteChildList()

sADMListEntry->CreateTime()

Create a timer

ADMTimerRef ASAPI (*CreateTime)(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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inMilliseconds</code></td>
<td>Number of milliseconds in the delay.</td>
</tr>
<tr>
<td><code>inAbortMask</code></td>
<td>Actions that cause a timer abort. If an action occurs, <code>inTimerAbortProc</code> is called. Maskable actions are listed in <code>ADMTracker.h</code>. Type: <code>ADMActionMask</code> (see <code>ADMTypes.h</code>)</td>
</tr>
<tr>
<td><code>inTimerProc</code></td>
<td>A callback with the following signature (see <code>ADMListEntry.h</code>): <code>ASBoolean ADMListEntryTimerProc(ADMListEntryRef inEntry, ADMTimerRef inTimer);</code> Executed if the delay succeeds. Returns a boolean. If it returns <code>true</code>, then <code>inTimerProc</code> will be called again after <code>inMilliseconds</code>. If it returns <code>false</code> then <code>inTimerProc</code> will no longer be called.</td>
</tr>
<tr>
<td><code>inTimerAbortProc</code></td>
<td>A callback with the following signature (see <code>ADMListEntry.h</code>): <code>ADMListEntryTimerAbortProc(ADMListEntryRef inEntry, ADMTimerRef inTimer, ADMAction inAbortAction);</code> Called if an action specified by <code>inAbortMask</code> occurs. The values for <code>inAbortAction</code> are of type <code>ADMAction</code> and are listed in <code>ADMTracker.h</code>.</td>
</tr>
<tr>
<td><code>inOptions</code></td>
<td>Currently unused. Always pass 0 (zero).</td>
</tr>
</tbody>
</table>

Returns

The new `ADMTimerRef`.

See also

`sADMListEntry->AbortTimer()`
The ADM List Entry Suite

ADM List Entry Suite Functions

sADMListEntry->DefaultDraw()  
Call an ADM list entry’s default drawing function

void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>inDrawer</td>
<td>An ADM drawer.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMHierarchyList->SetDrawProc()  
sADMHierarchyList->SetDrawProcRecursive()

Example

```c
void doNothingDrawHandler(ADMListEntryRef inEntry, ADMDrawerRef inDrawer) {  
    sADMListEntry->DefaultDraw(inEntry, inDrawer);  
}
```

sADMListEntry->DefaultNotify()  
Call an ADM list entry’s default notification function

void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inEntry</strong></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><strong>inNotifier</strong></td>
<td>An ADM notifier.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- `sADMHierarchyList->SetNotifyProc()`
- `sADMHierarchyList->SetNotifyProcRecursive()`

Example

```c
void doNothingNotificationHandler(ADMListEntryRef inEntry, ADMNotifierRef inNotifier) {
    sADMLListEntry->DefaultNotify(inEntry, inNotifier);
    // Custom behavior would go here...
}
```

**sADMLListEntry->DefaultTrack()**  
Call an ADM list entry's default tracking function

```c
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

```c
ASBoolean doNothingTrackHandler(ADMListEntryRef entry, ADMTrackerRef tracker) {
    return sADMListEntry->DefaultTrack(inEntry, inTracker);
}
```

---

`sADMListEntry->DeleteChildList()`  
Delete a child list for a list entry

```c
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

```c
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

```c
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**

<table>
<thead>
<tr>
<th>inEntry</th>
<th>An ADM list entry.</th>
</tr>
</thead>
</table>

| 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

```c
void ASAPI (*EnableChildSelection)(ADMListEntryRef inEntry, ASBoolean inFlag);
```

**Description**

The EnableChildSelection() function sets whether or not the entries of a child list are selectable.
The ADM List Entry Suite

ADM List Entry Suite Functions

The ADM List Entry Suite

ADM List Entry Suite Functions

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.
The ADM List Entry Suite

ADM List Entry Suite Functions

---

**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->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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>outPictureResID</td>
<td>Resource ID for the picture. If the entry does not have a disabled picture, returns 0.</td>
</tr>
<tr>
<td>outPictureResName</td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMListEntry->SetDisabledPictureID()
sADMListEntry->GetSelectedPicture()
sADMListEntry->GetSelectedPictureID()
sADMListEntry->GetPicture()
sADMListEntry->GetPictureID()
sADMListEntry->GetDisabledPicture()

sADMListEntry->GetEntryItem()

Get the list entry’s item

ADMItemRef ASAPI (*GetEntryItem)(ADMListEntryRef inEntry);

Description

**Note:** This API is deprecated in ADM V2.8.

The GetEntryItem() gets inEntry’s item.
The ADM List Entry Suite

ADM List Entry Suite Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

Returns

`inEntry`'s item.

`sADMListEntry->GetExpandArrowLocalRect()`  
Get location of expand/collapse arrow

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>outLocalRect</code></td>
<td>Location of the expand/collapse arrow within the bounds rectangle. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

Returns

None.

`sADMListEntry->GetHierarchyDepth()`  
Get the depth of a hierarchy list entry

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

Returns

How far down the hierarchy list `inEntry` is located (0-based).
See also

sADMLListEntry->GetVisualHierarchyDepth()

---

sADMLListEntry->GetID()

Get the ID of an ADM list entry

\[
\text{ASInt32 ASAPI (*GetID)(ADMLListEntryRef inEntry);}
\]

**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**

sADMLListEntry->SetID()

---

sADMLListEntry->GetIndex()

Get the index of a list entry

\[
\text{ASInt32 ASAPI (*GetIndex)(ADMLListEntryRef inEntry);}
\]

**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**

sADMEntry->GetIndex()
sADMListEntry->GetItem()

Get the item reference for a list entry

```c
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

```c
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

```c
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.
The ADM List Entry Suite

**ADM List Entry Suite Functions**

---

### Returns

None.

---

### `sADMLListEntry->GetPicture()`

**Gets a picture based on an icon**

```c
ADMIconRef ASAPI (*GetPicture)(ADMLListEntryRef inEntry);
```

**Description**

The `GetPicture()` function returns the picture used to draw `inEntry`.

**Parameters**

- `inEntry` An ADM list entry.

**Returns**

An ADM icon.

**See also**

- `sADMLListEntry->SetPicture()`
- `sADMLListEntry->GetPictureID()`
- `sADMLListEntry->GetSelectedPicture()`
- `sADMLListEntry->GetSelectedPictureID()`
- `sADMLListEntry->GetDisabledPicture()`
- `sADMLListEntry->GetDisabledPictureID()`

---

### `sADMLListEntry->GetPictureID()`

**Get the list entry's picture ID**

```c
void ASAPI (*GetPictureID)(ADMLListEntryRef 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.
Returns

None.

See also

sADMListEntry->SetPictureID()
sADMListEntry->GetPicture()
sADMListEntry->GetSelectedPicture()
sADMListEntry->GetSelectedPictureID()
sADMListEntry->GetDisabledPicture()
sADMListEntry->GetDisabledPictureID()

---

outPictureResID  Resource ID for the picture. If the entry does not have a picture, returns 0.

outPictureResName  Picture resource name.

---

sADMListEntry->GetSelectedPicture()

```
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

```c
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

```c
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`.

```c
void ASAPI (*GetText)(ADMListEntryRef inEntry, char* outText, ASInt32 inMaxLength);
```
### sADMListEntry->GetTextLength()

Get the length of the list entry’s text

```c
ASInt32 ASAPI (*GetTextLength)(ADMListEntryRef inEntry);
```

**Description**
The `GetTextLength()` function gets the number of characters in `inEntry`’s text.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry</td>
</tr>
</tbody>
</table>

**Returns**

Number of characters in `inEntry`’s text.

### sADMListEntry->GetUserData()

Get the user data of an ADM list entry

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry</td>
</tr>
</tbody>
</table>
The ADM List Entry Suite

ADM List Entry Suite Functions

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

ASIInt32 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>inHideEntry</td>
<td>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.</td>
</tr>
</tbody>
</table>

Returns

None.

sADMListEntry->Invalidate()

Invalidate the area of a list entry

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

Returns

true if inEntry is checked; false otherwise.
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()
sADMLListEntry->IsEntryNameHidden()

Determines whether a list entry’s name is visible or hidden

ASBoolean ASAPI (*IsEntryNameHidden)(ADMLListEntryRef inEntry);

Description

The IsEntryNameHidden() function determines whether the name of inEntry is visible or hidden.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

Returns

ture if the name of inEntry is hidden; false otherwise.

See also

sADMLListEntry->HideEntryName()

sADMLListEntry->IsHierarchyExpanded()

Determines whether a hierarchy list is expanded

ASBoolean ASAPI (*IsHierarchyExpanded)(ADMLListEntryRef inEntry);

Description

The IsHierarchyExpanded() function determines whether a hierarchy list for inEntry is expanded.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

Returns

true if the hierarchy list for inEntry is expanded; false otherwise.

sADMLListEntry->IsInBounds()

Determines whether a list entry is in bounds

ASBoolean ASAPI (*IsInBounds)(ADMLListEntryRef 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.

### sADMLListEntry->IsSelected()

Determines whether an ADM list entry is selected

```
ASBoolean ASAPI (*IsSelected)(ADMLListEntryRef 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 `sADMLListEntry->Select()` function.

**Parameters**

| `inEntry` | An ADM list entry. |

**Returns**

- **true** if `inEntry` selected; **false** otherwise.

**See also**

- `sADMLListEntry->IsChildSelectable()`  
- `sADMLListEntry->IsSelected()`  
- `sADMLListEntry->AreChildrenSelected()`  
- `sADMLListEntry->Select()`

### sADMLListEntry->IsSeparator()

Determine whether an ADM list entry is a separator line

```
ASBoolean ASAPI (*IsSeparator)(ADMLListEntryRef 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 `sADMLListEntry->MakeSeparator()` function.
The ADM List Entry Suite

ADM List Entry Suite Functions

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.
### sADMLListEntry->MakeInBounds()

**Description**

The `MakeInBounds()` function forces `inEntry` to be visible within a set boundary.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>A point in <code>inEntry</code>. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**Returns**

None.

### sADMLListEntry->MakeSeparator()

**Description**

The `MakeSeparator()` function makes `inEntry` into a separator. Menu lists can have separators that are used to divide their entries into categories.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inRect</code></td>
<td>A point in <code>inEntry</code>. Type: <code>ASRect</code> (see <code>ASTypes.h</code>)</td>
</tr>
</tbody>
</table>

**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.
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>ioPoint</td>
<td>A point in inEntry. Type: ASPoint (see ASTypes.h)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>
The ADM List Entry Suite

ADM List Entry Suite Functions

```
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

```c
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

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inNotifierType</code></td>
<td>Notifier type. The only standard notifier for ADM list entries is: <code>#define kADMUserChangedNotifier &quot;ADM User Changed Notifier&quot;</code></td>
</tr>
</tbody>
</table>

## Returns

None.

### ADMListEntry->SetBackgroundColor()

Set the list entry's background color

```c
void ASAPI (*SetBackgroundColor)(ADMListEntryRef inEntry, ADMColor inColor);
```

## Description

The `SetBackgroundColor()` function sets `inEntry`'s background color.

## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inColor</code></td>
<td>The color for the background. Type: <code>ADMColor</code> (see <code>ADMTypes.h</code>)</td>
</tr>
</tbody>
</table>

## Returns

None.

### sADMListEntry->SetDisabledPicture()

Set the list entry's disabled picture

```c
void ASAPI (*SetDisabledPicture)(ADMListEntryRef inEntry, ADMIconRef inPicture);
```

## Description

The `SetDisabledPicture()` function sets the picture that is used when `inEntry` is disabled.

## Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inPicture</code></td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>
Returns

None.

See also

\texttt{sADMListEntry->GetDisabledPicture()} \\
\texttt{sADMListEntry->SetDisabledPictureID()} \\
\texttt{sADMListEntry->SetPicture()} \\
\texttt{sADMListEntry->SetPictureID()} \\
\texttt{sADMListEntry->SetSelectedPicture()} \\
\texttt{sADMListEntry->SetSelectedPictureID()}

\texttt{sADMListEntry->SetDisabledPictureID()} \\

\begin{verbatim}
void ASAPI (*SetDisabledPictureID)(ADMListEntryRef inEntry, 
ASInt32 inPictureResID, const char* inPictureResName);
\end{verbatim}

Description

The \texttt{SetDisabledPictureID()} function sets the ID of the picture to be displayed for \texttt{inEntry} when it is disabled. \texttt{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

\begin{center}
\begin{tabular}{ll}
\textbf{inEntry} & An ADM entry. \\
\textbf{inPictureResID} & Resource ID for the picture. \\
\textbf{inPictureResName} & Picture resource name.
\end{tabular}
\end{center}

Returns

None.

See also

\texttt{sADMListEntry->GetDisabledPictureID()} \\
\texttt{sADMListEntry->SetDisabledPicture()} \\
\texttt{sADMListEntry->SetPicture()} \\
\texttt{sADMListEntry->SetPictureID()} \\
\texttt{sADMListEntry->SetSelectedPicture()} \\
\texttt{sADMListEntry->SetSelectedPictureID()}

\section*{Returns}

None.

\section*{See also}

\texttt{sADMListEntry->GetDisabledPicture()} \\
\texttt{sADMListEntry->SetDisabledPictureID()} \\
\texttt{sADMListEntry->SetPicture()} \\
\texttt{sADMListEntry->SetPictureID()} \\
\texttt{sADMListEntry->SetSelectedPicture()} \\
\texttt{sADMListEntry->SetSelectedPictureID()}

\section*{sADMListEntry->SetDisabledPictureID()}

Set the list entry's disabled picture ID

\begin{verbatim}
void ASAPI (*SetDisabledPictureID)(ADMListEntryRef inEntry, 
ASInt32 inPictureResID, const char* inPictureResName);
\end{verbatim}

Description

The \texttt{SetDisabledPictureID()} function sets the ID of the picture to be displayed for \texttt{inEntry} when it is disabled. \texttt{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

\begin{center}
\begin{tabular}{ll}
\textbf{inEntry} & An ADM entry. \\
\textbf{inPictureResID} & Resource ID for the picture. \\
\textbf{inPictureResName} & Picture resource name.
\end{tabular}
\end{center}

Returns

None.

See also

\texttt{sADMListEntry->GetDisabledPictureID()} \\
\texttt{sADMListEntry->SetDisabledPicture()} \\
\texttt{sADMListEntry->SetPicture()} \\
\texttt{sADMListEntry->SetPictureID()} \\
\texttt{sADMListEntry->SetSelectedPicture()} \\
\texttt{sADMListEntry->SetSelectedPictureID()}

\section*{Returns}

None.
**ADMLListEntry->SetDividingLineColor()**

Set the list entry’s dividing line color

```c
void ASAPI (*SetDividingLineColor)(ADMLListEntryRef 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>inColor</td>
<td>The color for the background. Type: ADMColor (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- ADMLListEntry->MakeSeparator()
- ADMLListEntry->IsSeparator()

---

**ADMLListEntry->SetEntryItem()**

Set the list entry’s item

```c
void ASAPI (*SetEntryItem)(ADMLListEntryRef inEntry, ADMItemRef inItem);
```

**Description**

**NOTE:** This API is deprecated in ADM V2.8.

The `SetEntryItem()` function sets `inEntry`’s item.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>inItem</td>
<td>The ADM item to associate with <code>inEntry</code>.</td>
</tr>
</tbody>
</table>

**Returns**

None.
sAMListEntry->SetEntryTextRect()

Set the list entry's edit field

```c
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

```c
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 `ADMTypes.h`)

**Returns**

None.

sADMListEntry->SetID()

Set the ID of an ADM list entry

```c
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.
The ADM List Entry Suite

ADM List Entry Suite Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>inEntryID</td>
<td>ID for inEntry.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMListEntry->GetID()

sADMListEntry->SetPicture()

Sets a picture based on an icon

```c
void ASAPI (*SetPicture)(ADMListEntryRef inEntry, ADMIconRef inPicture);```

Description

The `SetPicture()` function sets a picture based on an icon.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>inPicture</td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>

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

```c
void ASAPI (*SetPictureID)(ADMListEntryRef inEntry, ASInt32 inPictureResID, const char* inPictureResName);```
**Description**

The `SetPictureID()` function sets the ID for the picture to be displayed for `inEntry`. The `inPictureResID` is the ID of a platform picture or icon resource.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inPictureResID</code></td>
<td>Resource ID for the picture.</td>
</tr>
<tr>
<td><code>inPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

**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

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inPicture</code></td>
<td>An ADM icon.</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

- `sADMListEntry->GetSelectedPicture()`
- `sADMListEntry->SetSelectedPictureID()`
sADMListEntry->SetSelectedPictureID()

void ASAPI (*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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inPictureResID</code></td>
<td>Resource ID for the picture.</td>
</tr>
<tr>
<td><code>inPictureResName</code></td>
<td>Picture resource name.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

- sADMListEntry->GetSelectedPictureID()
- sADMListEntry->SetSelectedPicture()
- sADMListEntry->SetPicture()
- sADMListEntry->SetPictureID()
- sADMListEntry->SetDisabledPicture()
- sADMListEntry->SetDisabledPictureID()

sADMListEntry->SetText()

void ASAPI (*SetText)(ADMListEntryRef inEntry, const char* inText);

Description

The `SetText()` function sets `inEntry`'s text to the indicated C string.
The ADM List Entry Suite

ADM List Entry Suite Functions

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>inText</td>
<td>Text for inEntry.</td>
</tr>
</tbody>
</table>

Returns

None.

See also

sADMListEntry->GetText()

sADMListEntry->SetText Color() Set the list entry’s text color

```c
void ASAPI (*SetTextColor)(ADMListEntryRef inEntry, ADMColor inColor);
```

Description

The SetTextColor() function sets inEntry’s text color.

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inEntry</td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td>inColor</td>
<td>Text color for inEntry. Type: ADMColor (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

sADMListEntry->SetUserData() Set the user data of an ADM list entry

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inUserData</code></td>
<td>The 4-byte user value stored with <code>inEntry</code>. Type:</td>
</tr>
<tr>
<td></td>
<td><code>ADMUserData</code> (see ADMTypes.h)</td>
</tr>
</tbody>
</table>

Returns

None.

`sADMListEntry->Update()` Force an update of a list entry

```c
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

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.
sADMLListEntry->GetHelpID()  
Get the help ID of a list entry

```c
ASHelpID ASAPI (*GetHelpID)(ADMLListEntryRef inEntry);
```

**Description**

**NOTE:** This API is deprecated in ADM V2.8.

The `GetHelpID()` function gets the help ID for `inEntry`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

**Returns**

The help ID. Type: `ASHelpID` (See `ASHelp.h`)

**See also**

`sADMLListEntry->SetHelpID()`

---

sADMLListEntry->Help()  
Calls the help routine associated with a list entry

```c
void ASAPI (*Help)(ADMLListEntryRef inEntry);
```

**Description**

**NOTE:** This API is deprecated in ADM V2.8.

The `Help()` function calls the help for `inEntry`.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
</tbody>
</table>

**Returns**

None.

---

sADMLListEntry->SetHelpID()  
Set the help ID of a list entry

```c
void ASAPI (*SetHelpID)(ADMLListEntryRef 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>inEntry</code></td>
<td>An ADM list entry.</td>
</tr>
<tr>
<td><code>inHelpID</code></td>
<td>The resource ID for the ASHelp resource. Type: ASHelpID (See ASHelp.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

**See also**

`sADMEntry->GetHelpID()`
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:

```c
#define kADMNotifierSuite "ADM Notifier Suite"
```

with the version constant:

```c
#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:

```c
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 be 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:

```c
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:

```c
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:

```c
void ASAPI (*DefaultNotify)(ADMObjectRef inObject, ADMNotifierRef inNotifier);
```

You pass the `DefaultNotify()` function the arguments that were passed to your notifier function, for instance:

```c
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 the each of the ADM notifier functions:

```c
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

<table>
<thead>
<tr>
<th>Notifier</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>kADMUserChangedNotifier</td>
<td>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.</td>
</tr>
<tr>
<td>kADMBoundsChangedNotifier</td>
<td>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.</td>
</tr>
<tr>
<td>kADMEntryTextChangedNotifier</td>
<td>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.</td>
</tr>
<tr>
<td>kADMCloseHitNotifier</td>
<td>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.</td>
</tr>
<tr>
<td>kADMZoomHitNotifier</td>
<td>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.</td>
</tr>
<tr>
<td>kADMIntermediateChangedNotifier</td>
<td>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.</td>
</tr>
<tr>
<td>kADMCycleNotifier</td>
<td>This is received by ADM dialogs when a user is double clicking or triple clicking in the title bar of a tab palette.</td>
</tr>
</tbody>
</table>
### Table 17.1   ADM Notifier Types and Their Purposes

<table>
<thead>
<tr>
<th>Notifier</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>kADMCollapseNotifier</td>
<td>This is received by ADM dialogs when the user is collapsing the palette via the tab.</td>
</tr>
<tr>
<td>kADMExpandNotifier</td>
<td>This is received by ADM dialogs when the user is expanding the palette via the tab.</td>
</tr>
<tr>
<td>kADMGroupShowNotifier</td>
<td>This is received by an ADM item group when it is shown.</td>
</tr>
<tr>
<td>kADMGroupHideNotifier</td>
<td>This is received by an ADM item group when it is hidden.</td>
</tr>
<tr>
<td>kADMWindowDragMovedNotifier</td>
<td>This is received by ADM dialogs when the user moves the dialog by dragging it.</td>
</tr>
<tr>
<td>kADMContextMenuChangedNotifier</td>
<td>This is received by ADM dialogs when an edit operation has occurred via a clipboard operation.</td>
</tr>
<tr>
<td>kADMWindowShowNotifier</td>
<td>This is received by ADM dialogs when the user is in process of showing a window.</td>
</tr>
<tr>
<td>kADMWindowHideNotifier</td>
<td>This is received by ADM dialogs when the user is in the process of hiding a window.</td>
</tr>
<tr>
<td>kADMWindowActivateNotifier</td>
<td>This is received by ADM dialogs when a window is activated (focus moves into the dialog area).</td>
</tr>
<tr>
<td>kADMWindowDeactivateNotifier</td>
<td>This is received by ADM dialogs when a window is deactivated (focus moves to another screen area).</td>
</tr>
<tr>
<td>kADMNumberOutOfBoundsNotifier</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Text Item Notifiers:**
### ADM Notifier Types and Their Purposes

<table>
<thead>
<tr>
<th>Notifier</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>kADMPreClipboardCutNotifier</td>
<td>This is received by ADM text edit items when a user has issued a cut command, but the cut has not yet occurred.</td>
</tr>
<tr>
<td>kADMPostClipboardCutNotifier</td>
<td>This is received by ADM text edit items after the clipboard cut operation has occurred.</td>
</tr>
<tr>
<td>kADMPreClipboardCopyNotifier</td>
<td>This is received by ADM text edit items when a user has issued a copy command, but the copy has not yet occurred.</td>
</tr>
<tr>
<td>kADMPostClipboardCopyNotifier</td>
<td>This is received by ADM text edit items after the clipboard copy operation has occurred.</td>
</tr>
<tr>
<td>kADMPreClipboardPasteNotifier</td>
<td>This is received by ADM text items when a user has issued a paste command, but the paste has not yet occurred.</td>
</tr>
<tr>
<td>kADMPostClipboardPasteNotifier</td>
<td>This is received by ADM text edit items after the clipboard paste operation has occurred.</td>
</tr>
<tr>
<td>kADMPreClipboardClearNotifier</td>
<td>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.</td>
</tr>
<tr>
<td>kADMPostClipboardClearNotifier</td>
<td>This is received by ADM text edit items after the clipboard clear operation has occurred.</td>
</tr>
<tr>
<td>kADMPreTextSelectionChangedNotifier</td>
<td>This is received by ADM text edit items when user has issued a text selection change command but the change has not yet occurred.</td>
</tr>
<tr>
<td>kADMTextSelectionChangedNotifier</td>
<td>This is received by ADM text edit items after the text selection change operation has occurred.</td>
</tr>
</tbody>
</table>
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 \texttt{sADMItem->DefaultNotify()} function within your custom function,
even though in many cases there is no default notification. The default behaviors of item
notifiers include:

\begin{table}[h]
\centering
\begin{tabular}{|l|p{0.7\textwidth}|}
\hline
Notifier & Purpose \\
\hline
\texttt{kADMPreClipboardRedoNotifier} & This is received by ADM text edit items when the user has issued a redo command, but the redo has not yet occurred. \\
\hline
\texttt{kADMPostClipboardRedoNotifier} & This is received by ADM text edit items after a redo command has occurred. \\
\hline
\texttt{kADMPreClipboardUndoNotifier} & This is received by ADM text edit items when the user has issued an undo command, but the undo has not yet occurred. \\
\hline
\texttt{kADMPostClipboardUndoNotifier} & This is received by ADM text edit items after an undo command has occurred. \\
\hline
\end{tabular}
\caption{ADM Notifier Types and Their Purposes}
\end{table}
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()**

`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()**

`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()

Determines 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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inNotifier</strong></td>
<td>An ADM notifier.</td>
</tr>
<tr>
<td><strong>inNotifierType</strong></td>
<td>Notifier type as constant (see ADMNotifier.h for all possible notifier types listed as constant).</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inNotifier</strong></td>
<td>An ADM notification.</td>
</tr>
<tr>
<td><strong>inSkip</strong></td>
<td>If true, next clipboard operation specified by <em>inNotifier</em> is skipped.</td>
</tr>
</tbody>
</table>
The ADM Notifier Suite

ADM Notifier Suite Functions

Returns

None.
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:

```c
#define kADMTrackerSuite "ADM Tracker Suite"
```

with the version constant:

```c
#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:

```c
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 sADMDiallog->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

```c
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

```c
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()
The ADM Tracker Suite

ADM Tracker Suite Functions

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inTracker</td>
<td>An ADM tracker.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inTracker</td>
<td>An ADM tracker.</td>
</tr>
</tbody>
</table>

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.
The current mouse state. The available states include (see ADMTracker.h):

```c
typedef enum {
    kADMMouseNormal,
    kADMMouseCaptured,
    kADMMouseUncaptured
} ADMMouseState;
```

*sADMTracker->GetPoint()*

Get the point coordinates of the cursor

```c
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>inTracker</em></td>
<td>An ADM tracker.</td>
</tr>
<tr>
<td><em>outPoint</em></td>
<td>The cursor position when <em>inTracker</em> was created. Type: <em>ASPoint</em> (see ASTypes.h)</td>
</tr>
</tbody>
</table>

**Returns**

None.

*sADMTracker->GetTime()*

Get the time at which an action occurred.

```c
ADMTime ASAPI (*GetTime)(ADMTrackerRef inTracker);
```

**Description**

The *GetTime()* function gets the time at which an action occurred.

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>inTracker</em></td>
<td>An ADM tracker.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inTracker</td>
<td>An ADM tracker.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inTracker</td>
<td>An ADM tracker.</td>
</tr>
</tbody>
</table>

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 `ADMTracker.h`). 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 `ADMTracker.h`). This parameter is returned by `sADMTracker->GetModifiers()`.

Returns

- `true` if `inModifier` is pressed; `false` otherwise.

See also

- `sADMTracker->GetModifiers()`
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 should provide the supporting files you will need to work with your host application. The core ADM files are listed in Table A.1.

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM.txt</td>
<td>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.</td>
</tr>
<tr>
<td>ADMBasic.h</td>
<td>ADM Basic Suite functions</td>
</tr>
<tr>
<td>ADMDialog.h</td>
<td>ADM Dialog Suite functions</td>
</tr>
<tr>
<td>ADMDialogGroup.h</td>
<td>ADM DialogGroup Suite functions</td>
</tr>
<tr>
<td>ADMDrawer.h</td>
<td>ADM Drawer Suite functions</td>
</tr>
<tr>
<td>ADMEntry.h</td>
<td>ADM Entry Suite functions</td>
</tr>
<tr>
<td>ADMHierarchyList.h</td>
<td>ADM HierarchyList Suite functions</td>
</tr>
<tr>
<td>ADMIcon.h</td>
<td>ADM Icon Suite functions</td>
</tr>
<tr>
<td>ADMImage.h</td>
<td>ADM Image Suite functions</td>
</tr>
<tr>
<td>ADMItem</td>
<td>ADM Item Suite functions</td>
</tr>
<tr>
<td>ADMList.h</td>
<td>ADM List Suite functions</td>
</tr>
<tr>
<td>ADMListEntry.h</td>
<td>ADM List Entry Suite functions</td>
</tr>
<tr>
<td>ADMNotifier.h</td>
<td>ADM Notifier Suite functions</td>
</tr>
<tr>
<td>ADMResource.h</td>
<td>ADM Resource ID lists</td>
</tr>
<tr>
<td>ADMTracker.h</td>
<td>ADM Tracker Suite functions</td>
</tr>
<tr>
<td>ADMTypes.h</td>
<td>ADM resource, units, font, color, etc. definitions</td>
</tr>
</tbody>
</table>
The ADM folder of an SDK also contains other folders and files of interest, as shown in Table A.2.

<table>
<thead>
<tr>
<th>Folder</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IADM</td>
<td>C++ interfaces, see Using the C++ Interfaces.</td>
</tr>
<tr>
<td>Legacy</td>
<td>Maintenance files</td>
</tr>
</tbody>
</table>
**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 exists 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 **ADMAactions**.

**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().
FAQ

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 `kADMIntermediateChangedNotification` 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?

`ADMTextStatic` 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?

```c
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.

```cpp
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.

```cpp
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'.

Adobe Dialog Manager Programmer's Guide and Reference

571
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: kADMHostControlsCursorID.

Regarding this routine:

```cpp
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 \texttt{kADMForegroundColor} used? Is there a foreground color that is stored even when you've switched to something like \texttt{kADMBackgroundColor}? If so, what is the proper way to save/restore the color state? Does calling \texttt{sADMDrawer->SetRGBColor()} implicitly switch you to \texttt{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:

\begin{verbatim}
  sADMDrawer->GetRGBColor()
  sADMDrawer->SetRGBColor()
  sADMDrawer->GetADMColor()
  sADMDrawer->SetADMColor()
\end{verbatim}

The \texttt{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 \texttt{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:

\begin{verbatim}
  void ASAPI (*SkipNextClipboardOperation)(ADMNotifierRef notifier, ASBoolean skip);
\end{verbatim}

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 \url{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:

```c
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 workaround 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:

```c
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 DLOG 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 kADMModalDialogStyle, 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):

```c
IADMEntry 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.

```c
IADMEntry 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 `kADMIIntermediateChangedNotifier` 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
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);
ADMHierarchyList->SetInitProc(listRef, OnListEntryInit);
```

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 \texttt{bErase} flag set to \texttt{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.

\begin{verbatim}
scrollbarItem = sADMItem->GetChildItem(listItem, kADMListBoxScrollbarChildID);
sADMItem->Show(scrollbarItem, false);
\end{verbatim}
ADM Error Codes

The following error codes are returned wherever you see an `ASErr` returned by an API, or as the value of the `outError` 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'
- `kDockHostConflictsError` 'DOCK'
- `kTabGroupNotFoundErr` 'T!FD'
- `kAlreadyDockedError` 'DCKD'
- `kASDataStreamErr` 'DFER'
- `kASUnknownErr` 'UNK'
- `kASBufferTooSmallErr` 'BUFF'
- `kASMemoryErr` 'MEM'
ADM Error Codes
API Index

A
Abort
   sADMTracker 552
AbortTimer
   sADMDialog 147
   sADMEnergy 266
   sADMItem 381
   sADMListEntry 498
AboutBox
   sADMBasic 120
Activate
   sADMDialog 148
   sADMEnergy 266
   sADMItem 382
   sADMListEntry 499
AddItem
   sADMItem 382
AdjustItemTabOrder
   sADMDialog 148
ADMColorToRGBColor
   sADMBasic 133
AreChildrenSelected
   sADMListEntry 499
AreToolTipsEnabled
   sADMBasic 120
AreToolTipsSticky
   sADMBasic 121

B
Beep
   sADMBasic 121
BeginADMDrawer
   sADMImage 365
BeginAGMImageAccess
   sADMImage 366
BeginBaseAddressAccess
   sADMImage 367

C
Check
   sADMEnergy 267
   sADMListEntry 500
ChooseColor
   sADMBasic 122
Clear
   sADMDrawer 223
ClearRect
   sADMDrawer 223
CountDialogs
   sADMDialogGroup 210
Create
   sADMDialog 149
   sADMDrawer 224
   sADMEnergy 267
   sADMIcon 358
   sADMImage 367
   sADMItem 383
   sADMListEntry 501
CreateBitmap
   sADMImage 368
CreateChildList
   sADMListEntry 501
CreateFromImage
   sADMIcon 359
CreateGroupInDialog
   sADMDialog 152
CreateItem
   sADMDialog 153
CreateMenu
   sADMBasic 143
CreateOffscreen
   sADMImage 368
CreateTimer
   sADMDialog 154
   sADMEnergy 268
   sADMItem 384
   sADMListEntry 502
<table>
<thead>
<tr>
<th>Function Name</th>
<th>Type</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultDraw</td>
<td>sADMDialog</td>
<td>155</td>
</tr>
<tr>
<td>sADMItem</td>
<td>269</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td>385</td>
<td></td>
</tr>
<tr>
<td>sADMListEntry</td>
<td>503</td>
<td></td>
</tr>
<tr>
<td>DefaultFloatToText</td>
<td>sADMItem</td>
<td>386</td>
</tr>
<tr>
<td>DefaultNotify</td>
<td>sADMDialog</td>
<td>156</td>
</tr>
<tr>
<td>sADMItem</td>
<td>270</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td>387</td>
<td></td>
</tr>
<tr>
<td>sADMListEntry</td>
<td>504</td>
<td></td>
</tr>
<tr>
<td>DefaultTextToFloat</td>
<td>sADMItem</td>
<td>387</td>
</tr>
<tr>
<td>DefaultTrack</td>
<td>sADMDialog</td>
<td>156</td>
</tr>
<tr>
<td>sADMEntry</td>
<td>271</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td>388</td>
<td></td>
</tr>
<tr>
<td>sADMListEntry</td>
<td>504</td>
<td></td>
</tr>
<tr>
<td>DeleteChildList</td>
<td>sADMListEntry</td>
<td>505</td>
</tr>
<tr>
<td>DeselectAll</td>
<td>sADMHierarchyList</td>
<td>302</td>
</tr>
<tr>
<td>Destroy</td>
<td>sADMDialog</td>
<td>157</td>
</tr>
<tr>
<td>sADMItem</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td>271</td>
<td></td>
</tr>
<tr>
<td>sADMicon</td>
<td>359</td>
<td></td>
</tr>
<tr>
<td>sADMLimage</td>
<td>369</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td>389</td>
<td></td>
</tr>
<tr>
<td>sADMListEntry</td>
<td>506</td>
<td></td>
</tr>
<tr>
<td>DestroyItem</td>
<td>sADMDialog</td>
<td>158</td>
</tr>
<tr>
<td>DestroyMenu</td>
<td>sADMBasic</td>
<td>143</td>
</tr>
<tr>
<td>DisplayAsModal</td>
<td>sADMDialog</td>
<td>159</td>
</tr>
<tr>
<td>DisplayAsPopupModal</td>
<td>sADMDialog</td>
<td>159</td>
</tr>
<tr>
<td>DisplayMenu</td>
<td>sADMBasic</td>
<td>144</td>
</tr>
<tr>
<td>DrawADMLImage</td>
<td>sADMItem</td>
<td>387</td>
</tr>
<tr>
<td>DrawAGMImage</td>
<td>sADMDrawer</td>
<td>226</td>
</tr>
<tr>
<td>DrawDownArrow</td>
<td>sADMDrawer</td>
<td>227</td>
</tr>
<tr>
<td>DrawIcon</td>
<td>sADMDrawer</td>
<td>227</td>
</tr>
<tr>
<td>DrawIconCentered</td>
<td>sADMDrawer</td>
<td>228</td>
</tr>
<tr>
<td>DrawLeftArrow</td>
<td>sADMDrawer</td>
<td>229</td>
</tr>
<tr>
<td>DrawLine</td>
<td>sADMDrawer</td>
<td>230</td>
</tr>
<tr>
<td>DrawOval</td>
<td>sADMDrawer</td>
<td>230</td>
</tr>
<tr>
<td>DrawPolygon</td>
<td>sADMDrawer</td>
<td>231</td>
</tr>
<tr>
<td>DrawRaisedRect</td>
<td>sADMDrawer</td>
<td>231</td>
</tr>
<tr>
<td>DrawRecoloredIcon</td>
<td>sADMDrawer</td>
<td>232</td>
</tr>
<tr>
<td>DrawRecoloredResPicture</td>
<td>sADMDrawer</td>
<td>234</td>
</tr>
<tr>
<td>DrawRecoloredResPictureCentered</td>
<td>sADMDrawer</td>
<td>233, 235</td>
</tr>
<tr>
<td>DrawRect</td>
<td>sADMDrawer</td>
<td>236</td>
</tr>
<tr>
<td>DrawResPicture</td>
<td>sADMDrawer</td>
<td>236</td>
</tr>
<tr>
<td>DrawResPictureCentered</td>
<td>sADMDrawer</td>
<td>237</td>
</tr>
<tr>
<td>DrawRightArrow</td>
<td>sADMDrawer</td>
<td>238</td>
</tr>
<tr>
<td>DrawSunkenRect</td>
<td>sADMDrawer</td>
<td>239</td>
</tr>
<tr>
<td>DrawText</td>
<td>sADMDrawer</td>
<td>239</td>
</tr>
<tr>
<td>DrawTextCentered</td>
<td>sADMDrawer</td>
<td>240</td>
</tr>
<tr>
<td>DrawTextLabel</td>
<td>sADMDrawer</td>
<td>241</td>
</tr>
<tr>
<td>DrawTextLeft</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
sADMDrawer 242
DrawTextRight
  sADMDrawer 242
DrawUpArrow
  sADMDrawer 243

E
Enable
  sADMDialog 160
  sADMEntry 272
  sADMItem 390
  sADMListEntry 506
EnableChildSelection
  sADMListEntry 507
EnableTip
  sADMItem 467
EnableToolTips
  sADMBasic 122
EndADMDrawer
  sADMImage 370
EndAGMImageAccess
  sADMImage 369
EndBaseAddressAccess
  sADMImage 370
EndModal
  sADMDialog 160
ErrorAlert
  sADMBasic 123
ExpandHierarchy
  sADMListEntry 507

F
FillOval
  sADMDrawer 244
FillPolygon
  sADMDrawer 244
FillRect
  sADMDrawer 245
FindEntry
  sADMHierarchyList 302
  sADMList 475

G
GetAction
  sADMTracker 552
GetActiveEntry
  sADMHierarchyList 303
  sADMList 475
GetActiveLeafEntry
  sADMHierarchyList 303
GetADMColor
  sADMDrawer 245
GetADMWIndowPort
  sADMDrawer 246
GetAGMPort
  sADMDrawer 247
GetAllowMath
  sADMItem 390
GetAllowUnits
  sADMItem 391
GetAppFPS
  sADMBasic 134
GetAppUnits
  sADMBasic 134
GetBackColor
  sADMItem 391
GetBestSize
  sADMItem 392
GetBitsPerPixel
  sADMImage 371
GetBooleanValue
  sADMItem 392
GetBoundsRect
  sADMDialog 161
  sADMDrawer 247
  sADMEntry 273
  sADMItem 393
  sADMListEntry 508
GetByteWidth
  sADMImage 371
GetCancellItemID
  sADMDialog 162
GetCharacter
  sADMTracker 553
GetCheckGlyph
sADMEntry 273
GetChildItem
    sADMItem 393
GetClientRect
    sADMDrawer 248
GetCursorId
    sADMDialog 162
    sADMItem 395
GetData
    sADMIcon 360
GetDefaultIncrements
    sADMBasic 135
GetDefaultItemID
    sADMDialog 163
GetDestroyProc
    sADMDialog 163
    sADMHierarchyList 304
    sADMItem 395
    sADMList 476
GetDialog
    sADMItem 396
    sADMNotifier 547
GetDialogGroupInfo
    sADMDialogGroup 210
GetDialogTitle
    sADMDialog 164
    sADMDialogGroup 211
GetDialogStyle
    sADMDialog 164
GetDisabledPicture
    sADMEntry 274
    sADMItem 396
    sADMListEntry 508
GetDisabledPictureID
    sADMEntry 274
    sADMItem 397
    sADMListEntry 509
GetDivided
    sADMHierarchyList 304
GetDrawMode
    sADMDrawer 248
GetDrawProc
    sADMDialog 165
    sADMHierarchyList 305
GetEntry
    sADMHierarchyList 305
    sADMList 477
GetEntryHeight
    sADMHierarchyList 306
    sADMList 478
GetEntryItem
    sADMListEntry 510
GetEntryTextRect
    sADMHierarchyList 306
    sADMList 478
GetEntryWidth
    sADMHierarchyList 307
    sADMList 479
GetExpandArrowLocalRect
    sADMListEntry 510
GetExpandedIndex
    sADMHierarchyList 308
GetFixedValue
    sADMItem 398
GetFlags
    sADMHierarchyList 308
GetFloatToTextProc
    sADMItem 398
GetFloatValue
    sADMItem 399
GetFont
    sADMDialog 165
    sADMDrawer 249
    sADMItem 399
GetFontInfo
    sADMDrawer 249
GetForeColor
    sADMItem 400
GetFromResource
    sADMIcon 360
GetGlobalLeftMargin
    sADMHierarchyList 309
GetHasRollOverProperty
    sADMHierarchyList 309
GetHeight
    sADMItem 400
    sADMIcon 361
sADMImage 372
GetHelpID
   sADMDialog 207
   sADMEntry 296
   sADMItem 467
   sADMListEntry 537
GetHierarchyDepth
   sADMListEntry 511
GetHierarchyList
   sADMItem 401
GetHorizontalIncrement
   sADMDialog 166
GetID
   sADMDialog 167
   sADMEntry 275
   sADMItem 401
   sADMListEntry 511
GetIndentationWidth
   sADMHierarchyList 309
GetIndex
   sADMEntry 275
   sADMListEntry 512
GetIndexString
   sADMBasic 118
GetInitProc
   sADMHierarchyList 310
   sADMList 479
GetIntValue
   sADMItem 402
GetItem
   sADMDialog 167
   sADMHierarchyList 310
   sADMList 480
   sADMListEntry 508, 512
   sADMNotifier 548
GetItemStyle
   sADMItem 402
GetItemType
   sADMItem 402
GetJustify
   sADMItem 403
GetLargeIncrement
   sADMItem 403
GetLastADMError
   sADMBasic 136
GetLeafIndex
   sADMHierarchyList 311
GetList
   sADMEntry 276
   sADMItem 404
   sADMListEntry 512
GetLocalLeftMargin
   sADMHierarchyList 311
GetLocalRect
   sADMDialog 168
   sADMEntry 276
   sADMHierarchyList 312
   sADMItem 405
   sADMListEntry 513
GetMask
   sADMDialog 168
   sADMHierarchyList 312
   sADMItem 405
   sADMList 480
GetMaxFixedValue
   sADMItem 406
GetMaxFloatValue
   sADMItem 406
GetMaxHeight
   sADMDialog 169
GetMaxIntValue
   sADMItem 406
GetMaxTextLength
   sADMItem 407
GetMaxWidth
   sADMDialog 169
GetMenuID
   sADMHierarchyList 312
   sADMList 480
GetMinFixedValue
   sADMItem 407
GetMinFloatValue
   sADMItem 408
GetMinHeight
   sADMDialog 170
GetMinIntValue
   sADMItem 408
GetMinWidth
API Index

sADMDialog 170
GetModifiers
  sADMTracker 553
GetMouseState
  sADMTracker 553
GetNamedDialog
  sADMDialogGroup 211
GetNextItem
  sADMDialog 171
GetNonLeafEntryTextRect
  sADMHierarchyList 313
GetNonLeafEntryWidth
  sADMHierarchyList 313
GetNotifierData
  sADMDialog 172
  sADMHierarchyList 314
  sADMItem 409
  sADMList 481
GetNotifierType
  sADMNotifier 548
GetNotifyProc
  sADMDialog 172
  sADMHierarchyList 314
  sADMItem 409
  sADMList 481
GetNthDialog
  sADMDialogGroup 212
GetNumbersArePoints
  sADMBasic 136
GetOrigin
  sADMDrawer 250
GetPaletteLayoutBounds
  sADMBasic 137
GetParentEntry
  sADMHierarchyList 315
GetPicture
  sADMEntry 277
  sADMItem 410
  sADMListEntry 513
GetPictureID
  sADMEntry 277
  sADMItem 411
  sADMListEntry 514
GetPixel

  sADMImage 372
GetPluginRef
  sADMDialog 173
  sADMItem 411
GetPoint
  sADMTracker 554
GetPopupDialog
  sADMItem 412
GetPortRef
  sADMDrawer 250
GetPrecision
  sADMItem 412
GetPreviousItem
  sADMDialog 173
GetResPictureBounds
  sADMDrawer 251
GetRGBColor
  sADMDrawer 252
GetRolloverPicture
  sADMItem 413
GetRolloverPictureID
  sADMItem 413
GetScreenDimensions
  sADMBasic 137
GetSelectedPicture
  sADMEntry 278
  sADMItem 414
  sADMListEntry 514
GetSelectedPictureID
  sADMEntry 278
  sADMItem 415
  sADMListEntry 515
GetSelectionRange
  sADMItem 415
GetShowUnits
  sADMItem 416
GetSmallIncrement
  sADMItem 416
GetText
  sADMDialog 174
  sADMEntry 279
  sADMItem 417
  sADMListEntry 516
GetTextLength
<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sADMDialog</td>
<td>174</td>
<td></td>
</tr>
<tr>
<td>sADMEntry</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td>418</td>
<td></td>
</tr>
<tr>
<td>sADMListEntry</td>
<td>516</td>
<td></td>
</tr>
<tr>
<td>GetTextRectHeight</td>
<td>sADMDrawer</td>
<td>252</td>
</tr>
<tr>
<td>GetTextToFloatProc</td>
<td>sADMItem</td>
<td>418</td>
</tr>
<tr>
<td>GetTextWidth</td>
<td>sADMDrawer</td>
<td>253</td>
</tr>
<tr>
<td>GetThisFontInfo</td>
<td>sADMDrawer</td>
<td>253</td>
</tr>
<tr>
<td>GetTime</td>
<td>sADMTracker</td>
<td>554</td>
</tr>
<tr>
<td>GetTipString</td>
<td>sADMItem</td>
<td>468</td>
</tr>
<tr>
<td>GetTipStringLength</td>
<td>sADMItem</td>
<td>468</td>
</tr>
<tr>
<td>GetToolTipDelays</td>
<td>sADMBasic</td>
<td>123</td>
</tr>
<tr>
<td>GetTrackProc</td>
<td>sADMDialog</td>
<td>175</td>
</tr>
<tr>
<td>GetTrackProc</td>
<td>sADMHierarchyList</td>
<td>315</td>
</tr>
<tr>
<td>GetTrackProc</td>
<td>sADMItem</td>
<td>418</td>
</tr>
<tr>
<td>GetTrackProc</td>
<td>sADMList</td>
<td>482</td>
</tr>
<tr>
<td>GetType</td>
<td>sADMIcon</td>
<td>362</td>
</tr>
<tr>
<td>GetUnits</td>
<td>sADMItem</td>
<td>419</td>
</tr>
<tr>
<td>GetUpdateRect</td>
<td>sADMDrawer</td>
<td>254</td>
</tr>
<tr>
<td>GetUserData</td>
<td>sADMDialog</td>
<td>175</td>
</tr>
<tr>
<td>GetUserData</td>
<td>sADMEntry</td>
<td>280</td>
</tr>
<tr>
<td>GetUserData</td>
<td>sADMItem</td>
<td>419</td>
</tr>
<tr>
<td>GetUserData</td>
<td>sADMList</td>
<td>483</td>
</tr>
<tr>
<td>GetUserData</td>
<td>sADMListEntry</td>
<td>517</td>
</tr>
<tr>
<td>GetVerticalIncrement</td>
<td>sADMDialog</td>
<td>176</td>
</tr>
<tr>
<td>GetVirtualKey</td>
<td>sADMTracker</td>
<td>555</td>
</tr>
<tr>
<td>GetVisualHierarchyDepth</td>
<td>sADMListEntry</td>
<td>517</td>
</tr>
<tr>
<td>GetWantsFocus</td>
<td>sADMItem</td>
<td>420</td>
</tr>
<tr>
<td>GetWidth</td>
<td>sADMIcon</td>
<td>362</td>
</tr>
<tr>
<td>GetWidth</td>
<td>sADMImage</td>
<td>373</td>
</tr>
<tr>
<td>GetWidth</td>
<td>sADMBasic</td>
<td>138</td>
</tr>
<tr>
<td>GetWindowRef</td>
<td>sADMDialog</td>
<td>176</td>
</tr>
<tr>
<td>GetWindowRef</td>
<td>sADMItem</td>
<td>420</td>
</tr>
<tr>
<td>GetWorkspaceBounds</td>
<td>sADMBasic</td>
<td>138</td>
</tr>
<tr>
<td>GetWorkspaceBounds</td>
<td>sADMHierarchyList</td>
<td>317</td>
</tr>
<tr>
<td>GlobalToLocalPoint</td>
<td>sADMHierarchyList</td>
<td>317</td>
</tr>
<tr>
<td>GlobalToLocalRect</td>
<td>sADMHierarchyList</td>
<td>317</td>
</tr>
<tr>
<td>IgnoreForceRoman</td>
<td>sADMItem</td>
<td>421</td>
</tr>
<tr>
<td>IndexAllSelectedEntriesInHierarchy</td>
<td>sADMHierarchyList</td>
<td>318</td>
</tr>
<tr>
<td>IndexEntry</td>
<td>sADMHierarchyList</td>
<td>319</td>
</tr>
<tr>
<td>IndexEntry</td>
<td>sADMList</td>
<td>483</td>
</tr>
<tr>
<td>IndexExpandedEntry</td>
<td>sADMHierarchyList</td>
<td>319</td>
</tr>
<tr>
<td>IndexExpandedEntry</td>
<td>sADMList</td>
<td>483</td>
</tr>
<tr>
<td>IndexLeafEntry</td>
<td>sADMHierarchyList</td>
<td>320</td>
</tr>
<tr>
<td>IndexSelectedEntry</td>
<td>sADMHierarchyList</td>
<td>320</td>
</tr>
<tr>
<td>IndexSelectedEntry</td>
<td>sADMList</td>
<td>484</td>
</tr>
</tbody>
</table>
API Index

IndexUnNestedSelectedEntriesInHierarchy
  sADMHierarchyList 321
InsertEntry
  sADMHierarchyList 322
  sADMList 485
InsertGivenEntry
  sADMHierarchyList 322
IntersectClipPolygon
  sADMDrawer 254
IntersectClipRect
  sADMDrawer 255
Invalidate
  sADMDialog 177
  sADMEntry 281
  sADMHierarchyList 323
  sADMItem 422
  sADMListEntry 519
InvalidateRect
  sADMDialog 178
  sADMEntry 281
  sADMItem 422
InvertRect
  sADMDrawer 255
IsActive
  sADMDialog 178
  sADMEntry 282
  sADMItem 423
  sADMListEntry 520
IsChecked
  sADMEntry 282
  sADMListEntry 520
IsChildSelectable
  sADMListEntry 521
IsCollapsed
  sADMDialog 179
  sADMDialogGroup 212
IsDialogContextHidden
  sADMDialog 179
IsDockVisible
  sADMDialogGroup 213
IsEnabled
  sADMDialog 180
  sADMEntry 283
  sADMItem 423
  sADMListEntry 521
IsEntryNameHidden
  sADMListEntry 522
IsForcedOnScreen
  sADMDialog 180
IsFromResource
  sADMIcon 363
IsFrontTab
  sADMDialogGroup 214
IsHierarchyExpanded
  sADMListEntry 522
IsInBounds
  sADMEntry 283
  sADMListEntry 523
IsInRollOverState
  sADMItem 424
IsKnown
  sADMItem 424
IsNotifierType
  sADMNotifier 549
IsSelected
  sADMEntry 284
  sADMListEntry 523
IsSeparator
  sADMEntry 284
  sADMListEntry 524
IsStandAlonePalette
  sADMDialogGroup 214
IsTipEnabled
  sADMItem 470
isVisible
  sADMDialog 181
  sADMItem 425

K
Know
  sADMItem 426

L
LightweightErrorAlert
  sADMBasic 124
LoadToolTips
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>sADMDialog</td>
<td>182</td>
</tr>
<tr>
<td>LocalToGlobalPoint</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>323</td>
</tr>
<tr>
<td>LocalToGlobalRect</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>324</td>
</tr>
<tr>
<td>LocalToScreenPoint</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td>182</td>
</tr>
<tr>
<td>sADMEntry</td>
<td>285</td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>325</td>
</tr>
<tr>
<td>sADMItem</td>
<td>426</td>
</tr>
<tr>
<td>sADMListEntry</td>
<td>524</td>
</tr>
<tr>
<td>LocalToScreenRect</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td>183</td>
</tr>
<tr>
<td>sADMEntry</td>
<td>285</td>
</tr>
<tr>
<td>sADMItem</td>
<td>427</td>
</tr>
<tr>
<td>sADMListEntry</td>
<td>525</td>
</tr>
<tr>
<td>LookUpZString</td>
<td></td>
</tr>
<tr>
<td>sADMBasic</td>
<td>138</td>
</tr>
<tr>
<td><strong>M</strong></td>
<td></td>
</tr>
<tr>
<td>MakelnBounds</td>
<td></td>
</tr>
<tr>
<td>sADMEntry</td>
<td>286</td>
</tr>
<tr>
<td>sADMListEntry</td>
<td>525</td>
</tr>
<tr>
<td>MakeSeparator</td>
<td></td>
</tr>
<tr>
<td>sADMEntry</td>
<td>286</td>
</tr>
<tr>
<td>sADMListEntry</td>
<td>526</td>
</tr>
<tr>
<td>MessageAlert</td>
<td></td>
</tr>
<tr>
<td>sADMBasic</td>
<td>125</td>
</tr>
<tr>
<td>Modal</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td>183</td>
</tr>
<tr>
<td>Move</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td>185</td>
</tr>
<tr>
<td>sADMItem</td>
<td>427</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td></td>
</tr>
<tr>
<td>NumberOfAllSelectedEntriesInHierarchy</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>327</td>
</tr>
<tr>
<td>NumberOfEntries</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>325</td>
</tr>
<tr>
<td>sADMList</td>
<td>485</td>
</tr>
<tr>
<td>NumberOfExpandedEntries</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>328</td>
</tr>
<tr>
<td>NumberOfLeafEntries</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>326</td>
</tr>
<tr>
<td>NumberOfSelectedEntries</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>326</td>
</tr>
<tr>
<td>sADMList</td>
<td>486</td>
</tr>
<tr>
<td>NumberOfUnNestedSelectedEntriesInHierarchy</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>328</td>
</tr>
<tr>
<td><strong>P</strong></td>
<td></td>
</tr>
<tr>
<td>PickEntry</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>329</td>
</tr>
<tr>
<td>sADMList</td>
<td>486</td>
</tr>
<tr>
<td>PickLeafEntry</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>329</td>
</tr>
<tr>
<td>PluginAboutBox</td>
<td></td>
</tr>
<tr>
<td>sADMBasic</td>
<td>126</td>
</tr>
<tr>
<td>PopupModal</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td>185</td>
</tr>
<tr>
<td><strong>Q</strong></td>
<td></td>
</tr>
<tr>
<td>QuestionAlert</td>
<td></td>
</tr>
<tr>
<td>sADMBasic</td>
<td>126</td>
</tr>
<tr>
<td><strong>R</strong></td>
<td></td>
</tr>
<tr>
<td>RegisterItemType</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td>186</td>
</tr>
<tr>
<td>ReleaseADMWindowPort</td>
<td></td>
</tr>
<tr>
<td>sADMDrawer</td>
<td>256</td>
</tr>
<tr>
<td>ReleaseMouseCapture</td>
<td></td>
</tr>
<tr>
<td>sADMTracker</td>
<td>555</td>
</tr>
<tr>
<td>RemoveEntry</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td>330</td>
</tr>
<tr>
<td>sADMList</td>
<td>487</td>
</tr>
<tr>
<td>RemoveItem</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td>428</td>
</tr>
<tr>
<td><strong>S</strong></td>
<td></td>
</tr>
<tr>
<td>ScreenToLocalPoint</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td>187</td>
</tr>
<tr>
<td>sADMEntry</td>
<td>287</td>
</tr>
</tbody>
</table>
API Index

sADMHierarchyList 330
sADMItem 429
sADMListEntry 526
ScreenToLocalRect
  sADMDialog 187
  sADMEntry 287
  sADMItem 429
  sADMListEntry 527
Select
  sADMEntry 288
  sADMListEntry 527
SelectAll
  sADMItem 430
SelectByText
  sADMList 487
SendNotify
  sADMDialog 188
  sADMEntry 288
  sADMItem 430
  sADMListEntry 528
SetADMColor
  sADMDrawer 257
SetAlertButtonText
  sADMBasic 127
SetAllowMath
  sADMItem 431
SetAllowUnits
  sADMItem 431
SetAppFPS
  sADMBasic 139
SetAppUnits
  sADMBasic 139
SetBackColor
  sADMItem 432
SetBackgroundColor
  sADMHierarchyList 331
  sADMList 488
  sADMListEntry 528
SetBooleanValue
  sADMItem 432
SetBoundsRect
  sADMDialog 188
  sADMItem 433
SetCancelItemID
  sADMItem 434
  sADMListEntry 527
SetCheckGlyph
  sADMItem 289
SetClipPolygon
  sADMDrawer 257
SetClipRect
  sADMDrawer 258
SetCursorID
  sADMDialog 190
  sADMItem 435
SetDefaultIncrements
  sADMBasic 140
SetDefaultItemID
  sADMDialog 191
SetDestroyProc
  sADMDialog 191
  sADMHierarchyList 331
  sADMItem 435
  sADMList 488
SetDestroyProcRecursive
  sADMHierarchyList 332
SetDialogGroupInfo
  sADMDialogGroup 215
SetDialogName
  sADMDialog 192
SetDialogStyle
  sADMDialog 192
SetDisabledPicture
  sADMEntry 290
  sADMItem 435
  sADMListEntry 529
SetDisabledPictureID
  sADMEntry 290
  sADMItem 436
  sADMListEntry 529
SetDivided
  sADMHierarchyList 333
SetDividedRecursive
  sADMHierarchyList 334
SetDividingLineColor
  sADMListEntry 530
SetDrawMode
  sADMDrawer 258
SetDrawProc
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>sADMList</td>
<td></td>
</tr>
<tr>
<td>SetDrawProcRecursive</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>sADMList</td>
<td></td>
</tr>
<tr>
<td>SetEntryHeight</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>sADMList</td>
<td></td>
</tr>
<tr>
<td>SetEntryHeightRecursive</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>SetEntryItem</td>
<td></td>
</tr>
<tr>
<td>sADMListEntry</td>
<td></td>
</tr>
<tr>
<td>SetEntryTextRect</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>sADMList</td>
<td></td>
</tr>
<tr>
<td>SetEntryTextRectRecursive</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>SetEntryWidth</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>sADMList</td>
<td></td>
</tr>
<tr>
<td>SetEntryWidthRecursive</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>SetFixedValue</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetFlags</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>SetFlagsRecursive</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>SetFloatToTextProc</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetFloatValue</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetFont</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td></td>
</tr>
<tr>
<td>sADMDrawer</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>sADMListEntry</td>
<td></td>
</tr>
<tr>
<td>SetForcedOnScreen</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td></td>
</tr>
<tr>
<td>SetForeColor</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetHasRollOverProperty</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetHelpID</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td></td>
</tr>
<tr>
<td>sADMEntry</td>
<td></td>
</tr>
<tr>
<td>sADMListEntry</td>
<td></td>
</tr>
<tr>
<td>SetHorizontalIncrement</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetIndentationWidth</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>SetIndentationWidthRecursive</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>SetInitProc</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>sADMList</td>
<td></td>
</tr>
<tr>
<td>SetInitProcRecursive</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>SetIntValue</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetItemStyle</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetItemType</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetJustify</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetLargeIncrement</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetLocalLeftMargin</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>SetLocalRect</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>SetMask</td>
<td></td>
</tr>
<tr>
<td>sADMDialog</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>sADMList</td>
<td></td>
</tr>
<tr>
<td>SetMaskRecursive</td>
<td></td>
</tr>
<tr>
<td>sADMHierarchyList</td>
<td></td>
</tr>
<tr>
<td>sADMItem</td>
<td></td>
</tr>
<tr>
<td>sADMList</td>
<td></td>
</tr>
<tr>
<td>SetMaxFixedValue</td>
<td></td>
</tr>
</tbody>
</table>
API Index

- SetMaxFloatValue
  - sADMItem 446
- SetMaxHeight
  - sADMDialog 197
- SetMaxIntValue
  - sADMItem 447
- SetMaxTextLength
  - sADMItem 448
- SetMaxWidth
  - sADMDialog 197
- SetMenuID
  - sADMHierarchyList 346
  - sADMList 492
- SetMinFixedValue
  - sADMItem 448
- SetMinFloatValue
  - sADMItem 449
- SetMinHeight
  - sADMDialog 198
- SetMinIntValue
  - sADMItem 449
- SetMinWidth
  - sADMDialog 199
- SetNonLeafEntryTextRect
  - sADMHierarchyList 347
- SetNonLeafEntryTextRectRecursive
  - sADMHierarchyList 347
- SetNotifierData
  - sADMDialog 199
  - sADMHierarchyList 348
  - sADMItem 450
  - sADMList 493
- SetNotifyProc
  - sADMDialog 200
  - sADMHierarchyList 348
  - sADMItem 451
  - sADMList 494
- SetNotifyProcRecursive
  - sADMHierarchyList 349
- SetNumbersArePoints
  - sADMBasic 141
- SetOrigin
  - sADMDrawer 259
- SetPicture
  - sADMEntry 291
  - sADMItem 451
  - sADMListEntry 533
- SetPictureID
  - sADMEntry 292
  - sADMItem 452
  - sADMListEntry 533
- SetPixel
  - sADMidimage 373
- SetPlatformCursor
  - sADMBasic 119
- SetPluginRef
  - sADMItem 453
- SetPopupDialog
  - sADMItem 453
- SetPrecision
  - sADMItem 454
- SetRGBColor
  - sADMDrawer 260
- SetRolloverPicture
  - sADMItem 455
- SetRolloverPictureID
  - sADMItem 455
- SetSelectedPicture
  - sADMEntry 293
  - sADMItem 456
  - sADMListEntry 534
- SetSelectedPictureID
  - sADMEntry 293
  - sADMItem 457
  - sADMListEntry 534
- SetSelectionRange
  - sADMItem 458
- SetSmallIncrement
  - sADMItem 458
- SetTabGroup
  - sADMDialogGroup 215
- SetText
  - sADMDialog 201
  - sADMEntry 294
  - sADMItem 459
  - sADMListEntry 535
- SetTextColor
API Index

sADMListEntry 536
SetTextToFloatProc
sADMItem 461
SetTipString
sADMItem 471
SetTrackProc
sADMDialog 201
sADMHierarchyList 350
sADMItem 461
sADMList 494
SetTrackProcRecursive
sADMHierarchyList 351
SetUnits
sADMItem 462
SetUpdateEnabled
sADMDialog 202
SetUserData
sADMDialog 203
sADMEnergy 295
sADMHierarchyList 352
sADMItem 463
sADMList 495
sADMListEntry 536
SetVerticalIncrement
sADMDialog 203
SetWantsFocus
sADMItem 463
SetWindowRef
sADMDialog 204
Show
sADMDialog 205
sADMItem 464
ShowAllFloatingDialogs
sADMDialogGroup 216
ShowToolTip
sADMBasic 128
sADMItem 471
ShowUnits
sADMItem 464
Size
sADMDialog 205
sADMItem 465
SkipNextClipboardOperation
sADMMonifier 549
StandardGetDirectoryDialog
sADMBasic 131
StandardGetFileDialog
sADMBasic 129
StandardPutFileDialog
sADMBasic 132
StartMultipleItemInvalidate
sADMHierarchyList 353
StopMultipleItemInvalidate
sADMHierarchyList 353
StringToValue
sADMBasic 141
SubtractClipPolygon
sADMDriver 261
SubtractClipRect
sADMDriver 261
SwapEntries
sADMHierarchyList 354

T
TestAction
sADMTracker 556
TestModifier
sADMTracker 556
ToggleAllButNoCloseFloatingDialogs
sADMDialogGroup 216
ToggleAllFloatingDialogs
sADMDialogGroup 217

U
UnionClipPolygon
sADMDriver 262
UnionClipRect
sADMDriver 263
UnlinkEntry
sADMHierarchyList 354
UnregisterItemType
sADMDialog 206
Update
sADMDialog 206
sADMEnergy 295
sADMItem 465
API Index

sADMListEntry 537

V
ValueToString
sADMBasic 142

W
WasPercentageChange
sADMItem 466

Y
YesNoAlert
sADMBasic 132