Developing Operations Navigator plug-ins
Contents

Chapter 1. Developing Operations Navigator Plug-ins ........................................ 1

Chapter 2. What’s new for V5R1. ................................................................. 3

Chapter 3. Plug-in support in Operations Navigator ............................................. 5
  What you can do with a plug-in ...................................................... 5
  How plug-ins work ............................................................................ 6
  Plug-in requirements ........................................................................ 6
  Distribute plug-ins ........................................................................... 7
    Express setup file ........................................................................... 8
  Example: Information section of setup.ini ........................................... 8
  Example: Service section of setup.ini ............................................... 10
  Example: Identify files section of setup.ini ...................................... 10
  MRI setup file .................................................................................. 13
  Identifying plug-ins to Operations Navigator .................................... 13

Chapter 4. Install and run sample plug-ins ......................................................... 15
  Setting up sample C++ plug-ins ....................................................... 15
  Setting up sample Visual Basic plug-ins ............................................. 17
    Sample VB plug-in directory of files ............................................. 18
  Setting up the sample Java plug-in ................................................. 20
    Sample Java plug-in directory of files ......................................... 21

Chapter 5. Plug-in developers reference .......................................................... 25
  Operations Navigator structure and flow of control for C++ plug-ins ............... 25
  Operations Navigator COM interfaces for C++ ....................................... 26
    Description of IA4HierarchyFolder Interface .................................. 27
    IA4HierarchyFolder interface specifications listing ............................. 27
  Description of IA4PropSheetNotify interface ..................................... 37
    IA4PropSheetNotify interface specifications listing .......................... 37
  Operations Navigator API listing ......................................................... 39
    cwBUN_BindToLdapServer ............................................................... 43
    cwBUN_BindToLdapServerOnAs400 ............................................... 44
    cwBUN_CheckAS400Name ............................................................... 46
    cwBUN_CheckObjectAuthority ......................................................... 47
    cwBUN_CheckSpecialAuthority ....................................................... 49
    cwBUN_ConvertPidlToString .......................................................... 50
    cwBUN_EndODBCConnections ......................................................... 51
    cwBUN_FreeLdapBindInfo ............................................................... 52
    cwBUN_FreeLocalLdapServer ......................................................... 52
    cwBUN_GetAdminCacheState ......................................................... 53
    cwBUN_GetAdminCacheStateEx ....................................................... 54
    cwBUN_GetAdminValue ................................................................. 54
    cwBUN_GetAdminValueEx .............................................................. 55
    cwBUN_GetDisplayNameFromItemld .............................................. 57
    cwBUN_GetDisplayNameFromName ............................................... 58
    cwBUN_GetDisplayPathFromName ............................................... 58
    cwBUN_GetIconIndex ................................................................. 59
    cwBUN_GetIndexFromItemld .......................................................... 60
    cwBUN_GetIndexFromName ........................................................... 60
    cwBUN_GetIndexFromPidl ............................................................. 61
    cwBUN_GetLdapPublishCount ....................................................... 61
Chapter 1. Developing Operations Navigator Plug-ins

Are you interested in integrating your iSeries server administration tasks and client/server programs into a single application environment? The plug-in feature for Operations Navigator allows you to do just that! You can use plug-ins to consolidate third-party applications and specialized functions written in either C++, VB or Java into the Operations Navigator interface. Use these articles to learn what plug-ins are, how to create or customize them, and how to distribute them to your users.

Learn about plug-ins:

Plug-in support for Operations Navigator
Plan your plug-in by learning what plug-ins are, what you can do with them, and how to distribute them to your users.

Install and run the sample plug-in
The Client Access Express Toolkit helps you download and run sample plug-ins. You can use these samples to learn about plug-in support in Operations Navigator. Also, many developers use these samples as a base for their own modifications.

Develop plug-ins:

Plug-in developer’s reference
Find information about each type of plug-in’s architecture, and the flow of control within Operations Navigator. This topic also contains API listings, return codes, and links to ActiveX and COM information for C++ plug-ins, as well as links to the interfaces and classes relevant to Java plug-ins.

Distribute plug-ins
The Selective Setup feature in Client Access Express makes it easy to distribute the plug-in to your end users. Use this section to learn how to identify the new plug-in to Operations Navigator, and where to install the new plug-in.
Chapter 2. What’s new for V5R1

Developing Operations Navigator plug-ins shows how to consolidate third-party applications and specialized functions written in either C++, VB or Java into the Operations Navigator interface. Use these articles to learn what plug-ins are, how to create or customize them, and how to distribute them to your users.

New in the Information Center

For V5R1, these articles offer a description of plug-in support in Operations Navigator, descriptions of example plug-ins, a developers reference guide and a procedures for distributing the plug-in. See Developing Operations Navigator plug-ins for more information.

New for plug-ins

For V5R1, Java plug-ins support property sheets, and allow the integration of C++ and Java code in a single plug-in.
Chapter 3. Plug-in support in Operations Navigator

Plug-in support provides a convenient way to integrate your own functions and applications into a single user interface - Operation Navigator. These new functions and applications can vary in complexity from simple new behaviors to whole new applications. Regardless of what specific new ability your plug-in provides, integrating it into Operations Navigator provides several important benefits. For example, bundling common system tasks into a single location in Operations Navigator can dramatically simplify common administration and operation functions. Also, Operations Navigator’s GUI interface ensures that your integrated functions can be completed easily, and with only minimal prerequisite skills.

To help you plan your plug-in you may want to become familiar with the following topics:

- What you can do with a plug-in
  - Learn about what new functions you can add with a plug-in.

- How plug-ins work
  - Learn how plug-ins work by examining an example Java plug-in.

- Plug-in requirements
  - You can develop plug-ins in either C++, VB or Java. This topic describes the specific requirements for each language.

- Distributing plug-ins
  - You can easily distribute the new plug-in to your end users by placing it on the managing iSeries server. Client Access Express Selective Setup then detects the new plug-in and installs it on your client PCs.

What you can do with a plug-in

Plug-ins are sets of predefined classes and methods that Operations Navigator will invoke in response to a particular user action. You can use plug-ins to add or modify objects or folders in the Operations Navigator hierarchy that will represent your tools and applications. You can completely customize the support for your folders and objects by adding or modifying:

**Context menus**

Use context menus to launch applications, present new dialogs and add or modify behaviors.

**Property pages**

Use property pages to support customized attributes, i.e. additional security settings. You can add property pages to any object or folder that has a property sheet.

**Toolbars**

You can completely customize toolbars and buttons.

**Custom folders and objects**

You can add your own customized folders and objects into the Operations Navigator tree hierarchy.
How plug-ins work

The following illustration demonstrates how a Java plug-in that adds a new container to the Operations Navigator tree could work.

After identifying the new plug-in to the Windows registry, Operations Navigator will find the new plug-in and install it. Afterwards, the new container will appear in the Operations Navigator hierarchy. When the user selects the container, the plug-in’s Java code is called to obtain the container’s contents—in this case, a list of messages on the user’s default message queue.

**Operations Navigator dialog — messages in the message queue**

Operations Navigator communicates with the Java plug-in by invoking methods defined on a special Java interface: ListManager. This interface lets Java applications supply list data to the Navigator’s tree and list views. To integrate your application into Operations Navigator, you create a new Java class that implements this interface. The methods on the new class call into your existing Java application to obtain the list data, as shown below.

**How Operations Navigator calls an application to obtain list data**

What happens when the user wants to perform an action on one of your objects? The illustration below shows what happens when the user right-mouse clicks on a message object to display its context menu.

**Operations Navigator object context menu**

Operations Navigator calls a predefined method on another Java interface: ActionsManager. This interface obtains the list of menu items supported for message objects. Once again, you would create a new Java class that implements this interface. This is how you make your application’s specialized functions available to your users through Operations Navigator. When the user selects the menu item, the Navigator calls another ActionsManager method to perform the action. Your ActionsManager implementation calls your existing Java application, which then can display a confirmation dialog or some other more sophisticated user interface panel that allows the user to perform a specialized task. The Operations Navigator user interface is designed to let users work with lists of iSeries server resources and to perform actions on them. The architecture of the plug-in feature reflects this user interface design, both by defining interfaces for working with lists of objects in a hierarchy, and for defining actions on those objects. A third interface, DropTargetManager, handles drag-and-drop operations.

Plug-in requirements

Operations Navigator plug-in requirements differ according to the programming language that you use. However, all plug-ins require at least V3R1M3 of Client Access for Windows 95/NT, and V4R2 for OS/400. Visual Basic and Java plug-ins require Client Access Express V4R4 or greater.

**C++ plug-ins**

Plug-ins that are developed by using Microsoft’s Visual C++ programming language must be written using Version 4.2 or later.

**C++ plug-ins also require the following Operations Navigator APIs:**

<table>
<thead>
<tr>
<th>Header file</th>
<th>Import library</th>
<th>Dynamic Link Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>cwbun.h</td>
<td>cwbun.lib</td>
<td>cwbun.dll</td>
</tr>
<tr>
<td>cwbunpla.h (Application Administration APIs)</td>
<td>cwbapi.lib</td>
<td>cwbunpla.dll</td>
</tr>
</tbody>
</table>

**Java plug-ins**

All Java plug-ins require a small Windows resource DLL that contains certain information about your plug-in. This allows Operations Navigator to represent your function in the Navigator object hierarchy without having to load your plug-in’s implementation. The sample’s resource DLL was created by using Microsoft's Visual C++ version 4.2, but any C compiler that supports compiling and linking Windows resources may be used.

Operations Navigator provides a Java console as an aid to debugging. The console is activated by selecting a registry file to write the required console indicators to the Windows registry. When the console is activated, the JIT compiler is turned off to allow source code line numbers to appear in the stack trace, and any exceptions that are encountered in the Navigator’s Java infrastructure will be displayed in message boxes. The registry files for activating and for deactivating the console are provided with the sample Java plug-in.

The sample’s user interface was developed by using the Graphical Toolbox for Java, which is a part of the AS/400 Toolbox for Java. The Toolbox is an optionally installable component of Client Access Express, and may be installed by using Client Access Selective Setup.

**Visual Basic plug-ins**

Visual Basic plug-ins run on Version 5.0 of the Visual Basic runtime environment.

**Distribute plug-ins**

You can deliver your plug-in code to Operations Navigator users by including it with your OS/400 applications. The installation program for the application writes the plug-in’s code binaries, registry file, and translatable resources to a folder in the iSeries server Integrated File System (IFS). After completing this process, your end users can obtain the plug-in from the iSeries IFS (with the help of an AS/400 NetServer mapped network drive) by invoking the Client Access Selective Setup program. Selective Setup copies your plug-in code to the user’s machine, downloads the appropriate translatable resources based on the language settings on the user’s PC, and runs the registry file to write your plug-in’s registry information to the Windows registry.

**For this type of plug-in...**

<table>
<thead>
<tr>
<th>C++</th>
<th>Install in this directory...</th>
<th>And include these files...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>/QIBM/USERDATA/GUIPLUGIN/&lt;vendor&gt;.&lt;component&gt;</td>
<td>The registry file for the plug-in.</td>
</tr>
<tr>
<td>Or:</td>
<td>/QIBM/USERDATA/OpNavPlugin/&lt;vendor&gt;.&lt;component&gt;</td>
<td>The Express setup file on page 8 for the plug-in.</td>
</tr>
<tr>
<td>(To prevent installation without Client Access Express)</td>
<td></td>
<td>The ActiveX server DLL for the plug-in, and any associated code DLLs.</td>
</tr>
</tbody>
</table>

***Doug, what determines whether a plug-in will be compatible with Client Access? Will this be obvious to the plug-in developers? If not, we’ll need to explain.***

***Doug, will the plug-in developers know what <vendor>.<component> means?***
Java
/QIBM/USERDATA/OpNavPlugin/<vendor>.<component>
(Java plug-ins require Client Access Express)

- The registry file for the plug-in.
- The Express setup file for the plug-in.
- The Java JAR file containing all Java classes, HTML, .gif, PDML, PCML, and serialization files.

VB
/QIBM/USERDATA/OpNavPlugin/<vendor>.<component>
(VB plug-ins require Client Access Express)

- The registry file for the plug-in.
- The Express setup file for the plug-in.
- The ActiveX server DLL for the plug-in, and any associated code DLLs.

Note: The "<vendor>.<component>" subdirectory must match the one specified in the registry file.

Additionally, all plug-ins must create at least one directory below the <vendor>.<component> subdirectory called MRI29XX, where XX identifies a supported language. This directory should contain the correct national language version of the following items:
- The resource DLL for the plug-in
- The help files for the plug-in
- The MRI setup file for the plug-in

Upgrading or uninstalling the plug-in After the users have installed your new plug-in, you may choose either to upgrade it at a later date or ship bug fixes. When the code is upgraded on the iSeries server, the Client Access Express Check Version program will detect that this process has occurred and automatically downloads the updates to the users machines. Client Access also provides uninstall support, which lets your users completely remove the plug-in from their machines anytime they wish. Users can learn what plug-ins are installed on their machines by clicking on the Plug-ins tab on the Operations Navigator Properties for an iSeries server.

Restricting access to the plug-in with system policies and Application Administration If you provide a Windows policy template with your plug-in, you can also take advantage of Windows system policies to control which network users can install your plug-in. Additionally, you can use the iSeries server based Application Administration support in Operations Navigator to control which users and user groups can access your plug-in.

Express setup file
The Express setup file provides the Client Access Express Selective Setup program information necessary to install an Operations Navigator plug-in on a client workstation. It also provides information that allows the Client Access Express Login Service Check program to determine when the plug-in needs to be upgraded or serviced.

The file must be named SETUP.INI, and it must reside in the primary <vendor>.<component> directory for the plug-in on the iSeries server.

The format of the file conforms to that of a standard Windows configuration (.INI) file. The file is divided into three parts:
- Plug-in information [Example: Information section of setup.ini]
- Service [Example: Service section of setup.ini] on page 10
- Sections to identify the files [Example: Identify files section of setup.ini] on page 10 to install on the client workstation

Example: Information section of setup.ini
The first section of the Setup file (Plug-in Info) contains global information about the plug-in:
### Field in [Plugin Info] section of Setup.ini

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>English name of the plug-in. This name is displayed during installation of the plug-in when the translated name cannot be determined.</td>
</tr>
<tr>
<td>NameDLL</td>
<td>Name of the resource DLL that contains the translated name of the plug-in. This DLL is located in the MRI directories of the plug-in.</td>
</tr>
<tr>
<td>NameResID</td>
<td>Resource ID of the translated name in the MRI DLL. This field must contain the same value as the NameID field defined in the primary registry key for the plug-in.</td>
</tr>
<tr>
<td>Description</td>
<td>English description of the plug-in. This description is displayed during installation of the plug-in when the translated description cannot be determined.</td>
</tr>
<tr>
<td>DescriptionDLL</td>
<td>Name of the resource DLL that contains the translated description of the plug-in. This DLL is located in the MRI directories of the plug-in.</td>
</tr>
<tr>
<td>DescriptionResID</td>
<td>Resource ID of the translated description in the MRI DLL. This field must contain the same value as the DescriptionID field that is defined in the primary registry key for the plug-in.</td>
</tr>
<tr>
<td>Version</td>
<td>A numeric value that indicates the release level of the plug-in. The Client Access Login Service Check program uses this value to determine whether the plug-in needs to be upgraded on the client workstation. This value must be incremented by some amount for each new release of the plug-in. The Version value is compared to the current Version value of the installed plug-in that is on the client workstation. When this Version value is greater than the one already existing on the client workstation, the Client Access Login Service Check program upgrades the plug-in to the new Version.</td>
</tr>
<tr>
<td>VendorID</td>
<td>The &lt;VENDOR&gt;.&lt;COMPONENT&gt; string that is used to identify the plug-in. This string is used to create the registry key for the plug-in in the Client Access registry tree. The VendorID must be identical to the &lt;VENDOR&gt;.&lt;COMPONENT&gt; portion of the path where the plug-in will be installed on the iSeries server.</td>
</tr>
<tr>
<td>SupportExpress</td>
<td>SupportExpress is optional. This indicates that the plug-in is supported in Client Access Express, and that it will function correctly. If SupportExpress is set to NO or doesn’t exist, and the user selects to install this plug-in, a dialog box titled <strong>Operations Navigator Plug-in Not Supported</strong> will appear. This notifies you that you will be able to install the plug-in, but that it isn’t supported in Client Access Express. If you don’t want this dialog box to appear every time the plug-in is installed, and you know that the plug-in works with Client Access Express, then add SupportExpress and set it equal to YES.</td>
</tr>
<tr>
<td>JavaPlugin</td>
<td>JavaPlugin is used to indicate whether this is a Java plug-in. The install process needs to do some special processing if the plug-in is a Java plug-in. All JAR files must be installed into the \PLUGINS&lt;VENDOR&gt;.&lt;COMPONENT&gt; directory, and this value is used to determine whether the install process should do this. If the plug-in is a Java plug-in and this value is set to NO or doesn’t exist, the plug-in may not work after it is installed.</td>
</tr>
</tbody>
</table>
Example: Service section of setup.ini

The second section of the setup file (Service) provides the Client Access Login Service Check program with the information it requires to determine if a new fix level of the plug-in should be applied to the client workstation:

```
[Service]
FixLevel=0
AdditionalSize=0
```

Below is a listing of the meaning of each field:

<table>
<thead>
<tr>
<th>Field in [Service] section of Setup.ini</th>
<th>Description of field</th>
</tr>
</thead>
<tbody>
<tr>
<td>FixLevel</td>
<td>A numeric value that indicates the service level of the plug-in. The Client Access Login Service Check program uses this value to determine whether the plug-in requires servicing. This value must be incremented by some amount with each service release for a particular Version. The FixLevel value is compared to the current FixLevel value of the installed plug-in on the customer's computer. When this FixLevel value is greater than that of the plug-in that is installed on the client workstation, the Client Access Login Service Check program will Service the plug-in to the new FixLevel. The value must be reset to zero when a plug-in is upgraded to a new Version or release level.</td>
</tr>
<tr>
<td>AdditionalSize</td>
<td>The amount of DASD space that is required to store any new or additional executable files that will be added to the plug-in during servicing. Install uses this value to determine if the workstation has adequate disk space for the plug-in.</td>
</tr>
</tbody>
</table>

Example: Identify files section of setup.ini

The third and final portion of the setup file contains sections that identify the files that are to be installed on the client workstation. The section in which a file appears identifies the locations of the source and target for each file. These file sections are used during initial installations or during an upgrade to a new Version or release level.

The format for file entries in each file section should be `n=file.ext`, where `n` is the number of the file in that section. The numbering must start with one (1) and increment by one (1) until all of the files are listed in the section. For example:

```
[Base Files]
1=file1.dll
2=file2.dll
3=file3.dll
```

In all cases, only the file name and plug-in should be specified. Do not specify directory path names. If a file section contains no entries, the section simply is ignored.

**Note:** The Client Access Toolkit provides a sample setup file for three different sample plug-ins: C++, Java, and Visual Basic.

<table>
<thead>
<tr>
<th>Section in Setup.ini</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>[Base Files]</td>
<td>Files that are copied to \PLUGINS&lt;VENDOR&gt;.&lt;COMPONENT&gt; under the Client Access install directory. Normally, the ActiveX server DLL (and associated code DLLs) for the plug-in reside here. For <strong>C++ and Visual Basic</strong>, the ActiveX server DLL (and associated code DLLs) for the plug-in reside here. For <strong>Java</strong>, the Code JAR file name will reside here.</td>
</tr>
<tr>
<td>[Shared Files]</td>
<td>Files that are copied to the Client Access Shared directory.</td>
</tr>
<tr>
<td>[System Files]</td>
<td>Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory.</td>
</tr>
<tr>
<td>[Core Files]</td>
<td>Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory that are use counted in the registry and are never removed. These are typically re-distributable files.</td>
</tr>
<tr>
<td>[MRI Files]</td>
<td>Files that are copied from the MRI directories of the plug-in on the iSeries server to the CLIENT ACCESS\MRI29XX&lt;VENDOR&gt;.&lt;COMPONENT&gt; directories on the workstation. This typically is where the locale-dependent resources for a plug-in reside. This will include your Resource MRI DLL name.</td>
</tr>
<tr>
<td>[Java MRI29xx]</td>
<td>Java files that are copied from the MRI29xx directory of the plug-in on the iSeries server to the same directory to which the [Base Files] are installed. This typically is where the JAR MRI29xx resources for the plug-in reside. For each MRI29xx directory supported by the Java plug-in, there needs to be a [Java MRI29xx] section listing those files. This only is used by Java plug-ins.</td>
</tr>
<tr>
<td>[Help files]</td>
<td>The .HLP and .CNT files that are copied from the MRI directories of the plug-in on the iSeries server to the CLIENT ACCESS\MRI29XX&lt;VENDOR&gt;.&lt;COMPONENT&gt; directories on the workstation. The directory path to these files is written to HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WINDOWS\HELP in the Windows registry.</td>
</tr>
<tr>
<td>[Registry files]</td>
<td>The Windows registry file that is associated with the plug-in.</td>
</tr>
</tbody>
</table>
[Dependencies] Defines the sub components that must be installed before the plug-in can be installed. The values described below are optional. They are only needed if the plug-in requires other sub components to be installed besides the Operations Navigator base support sub component.

Two values are supported:

**AS400_Operations_Navigator**
- This value is used for legacy purposes to identify the sub components that must be installed if the plug-in is installed on Client Access V3R2M0. If the plug-in does not support running on Client Access V3R2M0, this value should not be specified.
- The sub components are specified in a comma-delimited list. A single sub component is specified as a single number (AS400_Operations_Navigator=3). The CWBUN.H header file contains a list of constants that are prefixed with CWBUN_OPNAV_. These constants provide the numeric values that are used in the comma-delimited list for AS400_Operations_Navigator.

**AS400_Client_Access_Express**
- This value is used to identify the sub components that must be installed if the plug-in is installed on Client Access Express.
- The sub components are specified in a comma-delimited list. A single sub component is specified as a single number (AS400_Client_Access_Express=3). The CWBAD.H header file contains a list of constants that are prefixed with CWBAD_COMP_. These constants provide the numeric values that are used in the comma-delimited list for AS400_Client_Access_Express. There are several CWBAD_COMP_ constants that identify PC5250 font sub components. These constants must not be used in the AS400_Client_Access_Express value and are listed below:

```
//5250 Display and Printer Emulator sub components
#define CWBAD_COMP_PC5250_BASE_KOREAN (150)
#define CWBAD_COMP_PC5250_PDPD_PDFPDT_KOREAN (151)
#define CWBAD_COMP_PC5250_BASE_SIMPCHIN (152)
#define CWBAD_COMP_PC5250_PDPD_PDFPDT_SIMPCHIN (153)
#define CWBAD_COMP_PC5250_BASE_TRAIDCHIN (154)
#define CWBAD_COMP_PC5250_PDPD_PDFPDT_TRAIDCHIN (155)
#define CWBAD_COMP_PC5250_BASE_STANDARD (156)
#define CWBAD_COMP_PC5250_PDPD_PDFPDT_STANDARD (157)
#define CWBAD_COMP_PC5250_FONT_ARABIC (158)
#define CWBAD_COMP_PC5250_FONT_BALTIC (159)
#define CWBAD_COMP_PC5250_FONT_LATIN2 (160)
#define CWBAD_COMP_PC5250_FONT_CYRILLIC (161)
#define CWBAD_COMP_PC5250_FONT_GREEK (162)
#define CWBAD_COMP_PC5250_FONT_HEBREW (163)
#define CWBAD_COMP_PC5250_FONT_LAO (164)
#define CWBAD_COMP_PC5250_FONT_THAI (165)
#define CWBAD_COMP_PC5250_FONT_TURKISH (166)
#define CWBAD_COMP_PC5250_FONT_VIET (167)
```
- This value is ignored by Client Access V3R2M0.

**Note:** Client Access Express will use the AS400_Client_Access_Express value if it exists. If it does not exist, it will use the AS400_Operations_Navigator value, if it exists. If neither value exists, then this section is ignored.

[Service Base Files] Files that are copied to \\PLUGINS\\<VENDOR>\\<COMPONENT> under the Client Access install directory.

[Service Shared Files] Files that are copied to the Client Access Shared directory.

[Service System Files] Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory.
MRI setup file

The MRI setup file provides the Client Access Selective Setup program with the information it needs to install the locale-dependent resources that are associated with an Operations Navigator plug-in on a client PC.

You must name the file MRISETUP.INI. A version of this file must reside in the MRI29XX subdirectory on the iSeries server for each national language that the plug-in supports.

The format of the file conforms to that of a standard Windows configuration (.INI) file. The file contains a single section, MRI Info. The MRI Info section provides the Version value for the MRI of the plug-in. The MRI for the plug-in includes all resource DLLs, as well as Help files (.HLP and .CNT) for a particular language. For example:

```
[MRI Info]
Version=0
```

The Client Access Selective Setup program checks the Version value of the MRI during an initial install and during an upgrade of the plug-in when incrementing the Version or release level of the plug-in. The MRI Version value in this file must match the Version value in the SETUP.INI file of the plug-in during the installation or upgrade. When these values do not match, the MRI files will not be copied to the client PC. The Client Access Toolkit provides a sample MRI setup file with the sample plug-in.

Identifying plug-ins to Operations Navigator

Plug-ins identify themselves to Operations Navigator by supplying information in the Windows registry when the plug-in software is installed on the Windows desktops of your users. The registry entries specify the location of the plug-in code and identify the classes that implement the special Operations Navigator interfaces. You can supply additional registry information that lets Operations Navigator determine whether the plug-in’s function should be activated for a particular iSeries system. For example, a plug-in may require a certain minimum release of OS/400, or it may specify that a certain product needs to be installed on the iSeries server in order for it to function.

When a user clicks on an iSeries server in the Operations Navigator hierarchy tree after installing a plug-in, Operations Navigator examines the iSeries server to determine whether it is capable of supporting the new plug-in. The software prerequisites (specified in the plug-in’s registry entries) are compared against the software installed on the iSeries server. If the plug-in’s requirements are satisfied, the new function will be displayed in the hierarchy tree. If the requirements are not met, the plug-in’s function will not appear for that iSeries server, unless the registry file specifies otherwise.
Chapter 4. Install and run sample plug-ins

The Client Access Express Toolkit supplies sample plug-ins in each of the supported programming languages. These samples provide an excellent way to learn how plug-ins work, and an efficient starting point for developing your own plug-ins. If you don’t already have the Client Access Express Toolkit installed, you will need to install it before working with any of the sample plug-ins. You can install the Toolkit through Client Access Selective Setup.

- Setting up the sample C++ plug-in
  Download the sample C++ plug-in and get it running in Operations Navigator.

- Setting up the sample VB plug-in
  Download the sample VB plug-in and get it running in Operations Navigator.

- Setting up the sample Java plug-in
  Download the sample Java plug-ins and get them running in Operations Navigator.

Note: Before starting to work on any of the sample plug-ins, you may want to be aware of the unique requirements for developing plug-ins in each of the three languages.

Setting up sample C++ plug-ins

This task involves building and running the sample ActiveX server DLL. The sample provides a functioning Developer Studio workspace that you can use to set breakpoints and to observe the behavior of a typical Operations Navigator plug-in. It also allows you to verify that your Developer Studio environment is set up correctly for compiling and linking plug-in code.

In order to get the sample C++ plug-in running or your PC, you must complete the following steps:

Download the C++ plug-in

![Download executable file](image-url) Download the executable file cppsmppq.exe. When you run the file it will extract all the files associated with the plug-in. Make a new directory, c:\MyProject, and copy all the files into it. If you create a different directory, you will have to modify registry file to specify the correct location for the plug-in.
Prepare to build an ActiveX server .dll

1. Create a new directory that is named “MyProject” on your local hard drive. This example assumes that the local drive is the C: drive.

   Note: If the new directory is not c:\MyProject, you will need to change the registry file.

2. Copy all of the sample files into this directory. You can download the samples from the Client Access Express Toolkit - Operations Navigator Plug-ins Web page.

3. In the Developer Studio, open the File menu and select Open Workspace.

4. In the Open Project Workspace dialog, switch to the MyProject directory and in Files of Type: select Makefiles (*.mak).

5. Select sampext.mak and click Open.

6. Open the Tools menu and select Options...

7. In the Directories tab, make sure that the Client Access Include directory appears at the top of your Include files search path.

8. In Show directories for:, select Library files. Make sure that the Client Access Lib directory appears at the top of your Library files search path.

9. Click OK to save the changes, then close and reopen Developer Studio. This is the only known way to force Developer Studio to save the search path changes to your hard disk.

Build the ActiveX server DLL

1. In the Developer Studio, open the Build menu and select Set Default Configuration...

2. In the Default Project Configuration dialog, select sampext Win32 Debug Configuration.

3. Open the Build menu and select Rebuild All to compile and link the DLL.

   Note: If the DLL does not compile and link cleanly, double-click the error messages in the Build window to locate and fix the errors. Then open the Build menu and select sampext.dll to restart the build.

Build the resource library

The resource DLL that contains the translatable text strings and other locale-dependent resources for the plug-in is included with the sample. This means that you do not have to create this DLL on your own. Even if your plug-in supports only one language, your plug-in code must load its text strings and locale-specific resources from this resource library.

To build the resource DLL, complete the following steps:

1. In Developer Studio, open the File menu and select Open Workspace... and select the MyProject directory.

2. Specify Makefiles (*.mak) in Files of Type:.

3. Select sampmri.mak and click Open.

4. Open the Build menu and select Rebuild All to compile and link the DLL.

Register the ActiveX server .dll

The SAMPDBG.REG file in the MyProject directory contains registry keys that communicate the location of the sample plug-in on your workstation to the Operations Navigator. If you specified a directory other than c:\MyProject, complete the following steps.

1. Open the SAMPDBG.REG file in the Developer Studio (or use your chosen text editor).

2. Replace all occurrences of “c:\\MyProject\\” with “x:\\<dir>\\,” where x is the drive letter where your directory resides and <dir> is the name of the directory.

3. Save the file.

4. In Windows Explorer, double-click the SAMPDBG.REG file. This will write the entries in the registry file to the Windows registry on your machine.

   Note: In Windows NT, you must login with administrative privileges on your workstation to write to the Windows registry.
Run Operations Navigator in the debugger

To run Operations Navigator and observe the sample plug-in in action, complete the following steps.

1. In Developer Studio, open the Build menu and select Debug —> Go.
2. At the prompt, type the fully-qualified path to the Operations Navigator executable in the Client Access Install directory on your workstation. The path will be C:\PROGRAM FILES\IBM\CLIENT ACCESS\CW BunNAv.EXE or something similar.
3. Click OK. The main window of the Operations Navigator will open.
4. Because you have just registered a new Navigator plug-in, a dialog in Operations Navigator will prompt you to scan for the new plug-in.
5. After the progress indicator finishes, click OK in the resulting dialog.
6. After the Navigator window refreshes, a new folder (3rd Party Sample Folder) appears in the hierarchy under the iSeries server that was initially selected. You can now interact with the plug-in in Operations Navigator and observe its behavior in the debugger.

Setting up sample Visual Basic plug-ins

The sample VB plug-in adds a folder to the Operations Navigator hierarchy that provides a list of OS/400 libraries, and illustrates how to implement properties and actions on those library objects.

In addition to installing the plug-in code, the sample plug-in includes a Readme.txt file, and to registry files, one for use during development, and another for distribution with the retail version. See the Sample VB plug-in directory of files for detailed description of all the files included with the VB plug-in.

In order to get the sample VB plug-in running or your PC, you must complete the following steps:

<table>
<thead>
<tr>
<th>Download the VB plug-in</th>
<th>Download the executable file vbopnav.exe. 🔄 When you run the file it will extract all the files associated with the plug-in. Make a new directory, c:\VB Sample, and copy all the files into it. If you create a different directory, you will have to modify registry file to specify the correct location for the plug-in.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build the ActiveX server DLL</td>
<td>Note: If either of these references do not appear in your References dialog, select Browse and look for cwbx.dll and cw bunvbii.dll in the Client Access Express shared The IBM Client Access Express ActiveX Object Library contains OLE automation objects that the sample application requires to make remote command calls to the iSeries server. The Operations Navigator Visual Basic Plug-in Support contains classes and interfaces required to create a Visual Basic Plug-in. directory.</td>
</tr>
<tr>
<td>Build the resource library</td>
<td>Select Make from the Visual Basic file menu to build the DLL. If it doesn’t compile and link, locate and fix the errors, and then rebuild the DLL.</td>
</tr>
<tr>
<td>Register the plug-in</td>
<td>Double-click the file vbsmpdbg.reg in order to register the plug-in. If you did not use the directory c:\VB Sample, edit the registry file, and replace all occurrences of “c:\VB Sample” with the fully-qualified path to the plug-in code. You must use double backslashes in the path.</td>
</tr>
</tbody>
</table>

Chapter 4. Install and run sample plug-ins  17
Run the plug-in in Operations Navigator, and click on the “+” next to an iSeries server to expand the tree. Operations Navigator will detect the changes to the registry, and prompt you to scan the iSeries server in order to verify that it is capable of supporting the new plug-in. After completing the scan, Operations Navigator will display the new plug-in in the tree hierarchy.

### Sample VB plug-in directory of files

The following tables describe all of the files included with the sample VB plug-in for v5r1.

<table>
<thead>
<tr>
<th>Visual Basic project file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbsample.vbp</td>
<td>Visual Basic 5.0 project file</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VB forms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authority.frm</td>
<td>Set authority form</td>
</tr>
<tr>
<td>delete.frm</td>
<td>Confirm delete form</td>
</tr>
<tr>
<td>propsht.frm</td>
<td>Property Sheet form</td>
</tr>
<tr>
<td>sysstat.frm</td>
<td>System status form</td>
</tr>
<tr>
<td>wizard.frm</td>
<td>Create new library wizard form</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VB Modules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>global.bas</td>
<td>Global declarations.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VB Class Modules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actnman.cls</td>
<td>SampleActions Manager class</td>
</tr>
<tr>
<td>dropman.cls</td>
<td>Sample Drop Target Manager class</td>
</tr>
<tr>
<td>library.cls</td>
<td>Library class</td>
</tr>
<tr>
<td>listman.cls</td>
<td>Sample List Manager class</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VB Binaries</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>authority.frx</td>
<td>Set authority form binary</td>
</tr>
<tr>
<td>delete.frx</td>
<td>Confirm delete form binary</td>
</tr>
<tr>
<td>propsht.frx</td>
<td>Property Sheet form binary</td>
</tr>
<tr>
<td>sysstat.frx</td>
<td>System status form binary</td>
</tr>
<tr>
<td>wizard.frx</td>
<td>Create new library wizard form binary</td>
</tr>
<tr>
<td>vbsample.bin</td>
<td>Vbsample binary</td>
</tr>
<tr>
<td>Configuration settings</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>mrisetup.ini</td>
<td>Install information for plug-in’s translatable resources</td>
</tr>
<tr>
<td>setup.ini</td>
<td>Install information for plug-in’s executables</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Registry entries</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbsmpdbg.reg</td>
<td>Registry file for use during development.</td>
</tr>
<tr>
<td>vbsmprls.reg</td>
<td>Registry file for use by Client Access during installation.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Files for constructing the resource DLL</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vbsmpmri.mak</td>
<td>Make File</td>
</tr>
<tr>
<td>vbsmpmri.rc</td>
<td>RC file</td>
</tr>
<tr>
<td>vbsmpres.h</td>
<td>Header file</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Images</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compass.bmp</td>
<td>Operations Navigator icon</td>
</tr>
<tr>
<td>lib.ico</td>
<td></td>
</tr>
<tr>
<td>vbsmpflr.ico</td>
<td>Visual Basic Sample plug-in folder in open and closed state.</td>
</tr>
<tr>
<td>vbsmplib.ico</td>
<td>Visual Basic Sample plug-in library icon.</td>
</tr>
</tbody>
</table>
### Setting up the sample Java plug-in

The sample Java plug-in works with message queues in QUSRYSYS on a given iSeries server. The first plug-in allows you to view, add and delete messages in your default message queue, the one with the same name as your iSeries user ID. The second plug-in adds support for multiple message queues. Finally, the third plug-in adds the ability to drag and drop messages between queues.

In addition to installing the plug-in code, the sample plug-in includes Java docs, a Readme.txt file, and two registry files, one for use during development and another for distribution with the retail version. See the Sample Java plug-in directory of files for a detailed description of all files included with the Java plug-ins.

To set up the sample Java plug-in:

1. **Download the sample Java plug-ins**
   - Download the executable file jvopnav.exe. When you run this file, it will extract all of the previously mentioned files. You should allow the executable to install the files in the default directory: `jvopnav\com\ibm\as400\opnav`.
2. **Identify the plug-in to Operations Navigator**
   - Edit the file `MsgQueueSampleX.reg` in `jvopnav\com\ibm\as400\opnav\MsgQueueSampleX`. (X=1, 2 or 3, depending on which sample you are installing.)
   - Find the lines: "NLS"="c:\jvopnav\win32\mri\MessageQueuesMRI.dll" and "JavaPath"="c:\jvopnav"
   - Replace "c:\" with the fully-qualified path to the jvopnav directory on your PC. You must double all back slashes in the path.
3. **Save your changes, and double click the registry file.**
4. **Run the sample Java plug-in.**
   1. Start Operations Navigator, and click on the "+" next to an iSeries server to expand the tree.
   2. Operations Navigator will detect the changes to the registry, and prompt you to scan the iSeries server in order to verify that it is capable of supporting the new plug-in.
   3. Click **Scan Now**
   4. Operations Navigator will scan the iSeries server. When it finishes, it will display a new folder in the hierarchy tree, Java Message Queue Sample 1, 2 or 3.
   5. Double click on the new folder
   6. The first sample plug-in will display the contents of your default message queue in QUSRYSYS on the iSeries server. The second and third samples will display a list of message queues.
   7. Add a new message by right-clicking on the message queue folder, and selecting **New -> Message**.
   8. The plug-in displays a PDML dialog allowing you to enter the message text.
   9. Delete a message by right-clicking on a message and selecting **Delete**. You can also do this from the toolbar.
   10. If you’re using the third sample plug-in, you can select a message, drag it to another queue, and then drop it.
   11. The plug-in will then move the message to the other queue.
## Sample Java plug-in directory of files

The following tables describe all of the files included with the sample Java plug-ins for v5r1. For more information, read the plug-in’s javadoc documentation. These were installed in your jvopnav\com\ibm\as400\opnav\MsgQueueSample1\docs directory. Start with the file Package-com.ibm.as400.opnav.MsgQueueSample1.html.

The sample’s package name is com.ibm.as400.opnav.MsgQueueSample1. All class names are prefixed with “Mq” to differentiate them from like-named classes in other packages.

<table>
<thead>
<tr>
<th>Java source code files; first sample plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MqMessagesListManager.java</td>
<td>The ListManager for lists of messages.</td>
</tr>
<tr>
<td>MqActionsManager.java</td>
<td>The ActionsManager implementation which handles all context menus for the plug-in.</td>
</tr>
<tr>
<td>MqMessageQueue.java</td>
<td>A collection of iSeries server message objects on a message queue.</td>
</tr>
<tr>
<td>MqMessage.java</td>
<td>An object representing an iSeries server message.</td>
</tr>
<tr>
<td>MqNewMessageBean.java</td>
<td>The UI DataBean implementation for the “New Message” dialog.</td>
</tr>
<tr>
<td>MqDeleteMessageBean.java</td>
<td>The UI DataBean implementation for the “Confirm Delete” dialog</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Java source code files; second sample plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MqListManager.java</td>
<td>The master ListManager implementation for the plug-in.</td>
</tr>
<tr>
<td>MqMessageQueuesListManager.java</td>
<td>A slave ListManager for lists of message queues.</td>
</tr>
<tr>
<td>MqMessagesListManager.java</td>
<td>A slave ListManager for lists of messages.</td>
</tr>
<tr>
<td>MqActionsManager.java</td>
<td>The ActionsManager implementation which handles all context menus for the plug-in.</td>
</tr>
<tr>
<td>MqMessageQueueList.java</td>
<td>A collection of iSeries server message queues.</td>
</tr>
<tr>
<td>MqMessageQueue.java</td>
<td>A collection of iSeries server message objects on a particular queue.</td>
</tr>
<tr>
<td>MqMessage.java</td>
<td>An object representing an iSeries server message.</td>
</tr>
<tr>
<td>MqNewMessageBean.java</td>
<td>The UI DataBean implementation for the “New Message” dialog.</td>
</tr>
<tr>
<td>MqDeleteMessageBean.java</td>
<td>The UI DataBean implementation for the “Confirm Delete” dialog.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Java source code files; third sample plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MqListManager.java</td>
<td>The master ListManager implementation for the plug-in.</td>
</tr>
<tr>
<td>MqMessageQueuesListManager.java</td>
<td>A slave ListManager for lists of message queues.</td>
</tr>
<tr>
<td>MqMessagesListManager.java</td>
<td>A slave ListManager for lists of messages.</td>
</tr>
<tr>
<td>MqActionsManager.java</td>
<td>The ActionsManager implementation which handles all context menus for the plug-in.</td>
</tr>
<tr>
<td>MqDropTargetManager.java</td>
<td>The DropTargetManager implementation which handles drag/drop for the plug-in.</td>
</tr>
<tr>
<td>MqMessageQueueList.java</td>
<td>A collection of iSeries server message queues.</td>
</tr>
</tbody>
</table>
### MqMessageQueue.java
A collection of iSeries server message objects on a particular queue.

### MqMessage.java
An object representing an iSeries server message.

### MqNewMessageBean.java
The UI DataBean implementation for the "New Message" dialog.

### MqDeleteMessageBean.java
The UI DataBean implementation for the "Confirm Delete" dialog.

### PDML files

<table>
<thead>
<tr>
<th>Description</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains all Java UI panel definitions for the plug-in.</td>
<td>MessageQueueGUI.pdml</td>
</tr>
<tr>
<td>The associated Java resource bundle (subclasses java.util.ListResourceBundle).</td>
<td>MessageQueueGUI.java</td>
</tr>
</tbody>
</table>

### Online help files

<table>
<thead>
<tr>
<th>Description</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Online help skeleton for the “New Message” dialog.</td>
<td>IDD_MSGQ_ADD.html</td>
</tr>
<tr>
<td>Online help skeleton for the “Confirm Delete” dialog.</td>
<td>IDD_MSGQ_CONFIRM_DELETE.html</td>
</tr>
</tbody>
</table>

### Serialized files

<table>
<thead>
<tr>
<th>Description</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serialized panel definition for the “New Message” dialog.</td>
<td>IDD_MSGQ_ADD.pdml.ser</td>
</tr>
<tr>
<td>Serialized panel definition for the “Confirm Delete” dialog.</td>
<td>IDD_MSGQ_CONFIRM_DELETE.pdml.ser</td>
</tr>
</tbody>
</table>

**Note:** If you make changes to MessageQueueGUI.pdml, rename these files. Otherwise your changes will not be reflected in the panels.

### Registry entries

<table>
<thead>
<tr>
<th>Description</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows registry entries that tell Operations Navigator that this plug-in exists, and identifies its Java interface implementation classes.</td>
<td>MsgQueueSample1.reg</td>
</tr>
<tr>
<td></td>
<td>MsgQueueSample2.reg</td>
</tr>
<tr>
<td></td>
<td>MsgQueueSample3.reg</td>
</tr>
<tr>
<td>The registry file for distribution with the retail version of your plug-in. This version of the registry file cannot be read directly by Windows. It contains substitution variables that represent the directory path of the Client Access Express installation directory. When the user invokes the Client Access Selective Setup program to install your plug-in from the iSeries server, Selective Setup reads this registry file, fills in the correct directory paths, and writes the entries to the registry on the user’s machine. The entries in this file should therefore be kept in sync with the registry file used in development.</td>
<td>MsgQueueSample1install.reg</td>
</tr>
<tr>
<td></td>
<td>MsgQueueSample2install.reg</td>
</tr>
<tr>
<td></td>
<td>MsgQueueSample3install.reg</td>
</tr>
</tbody>
</table>
Chapter 5. Plug-in developers reference

Operations Navigator handles plug-ins in each programming language uniquely. You can use the following topics to learn about the flow of control in Operations Navigator for each type of plug-in, as well as specific reference information regarding the unique interfaces for each language.

C++ Reference
- Flow of Control in Operations Navigator
- COM Interfaces
- API listing
- Return Codes

VB Reference
- Flow of control in Operations Navigator
- VB Interfaces

Java Reference
- Flow of control in Operations Navigator
- Java Classes and Interfaces

In addition to reference information specific to each language, each plug-in requires some customization to Windows registry files.

Plug-in registry files
After modifying the sample plug-ins, you’ll need to make some modifications to the registry files. This topic provides a walk-through of the registry files for each type of plug-in, and recommends some modifications.

Operations Navigator structure and flow of control for C++ plug-ins

The internal architecture of the Operations Navigator product reflects that it is intended to serve as an integration point for an extensible, broad-based operations interface for the iSeries server. Each functional component of the interface is packaged as an ActiveX server DLL. Operations Navigator uses Microsoft’s Component Object Model (COM) technology to activate only the component implementations that currently are needed to service a user request. This avoids the problem of having to load the entire product at start up, thereby consuming the majority of Windows resources, and impacting performance of the entire system. Multiple servers may register their request to add menu items and dialogs to a given object type in the Navigator hierarchy.

Plug-ins work by responding to method calls from Operations Navigator that are generated in response to user actions. For example, when a user right-clicks on an object in the Navigator hierarchy, the Navigator constructs a context menu for the object, and displays the menu on the screen. The Navigator obtains the menu items by calling each plug-in that has registered its intention to supply context menu items for the selected object type.

The functions that are implemented by a plug-in logically are grouped into “interfaces.” An interface is a set of logically related methods on a class that Operations Navigator can call to perform a specific function. The Component Object Model supports the definition of interfaces in C++ through the declaration of an abstract class that defines a set of pure virtual functions. Classes that call the interface are known as implementation classes. Implementation classes subclass the abstract class definition and provide C++ code for each of the functions defined on the interface.
A given implementation class may implement as many interfaces as the developer chooses. When creating a new project workspace for an ActiveX server DLL in the Developer Studio, the AppWizard generates macros that facilitate interface implementation. Each interface is declared as a nested class on a containing implementation class. The nested class has no member data and does not use any functions other than those that are defined on its interface. Its methods typically call functions on the implementation class to get and set state data, and to perform the actual work that is defined by the interface specification.

### Operations Navigator COM interfaces for C++

The functions implemented by a plug-in logically are grouped into **Component Object Model (COM) interfaces**. An interface is a set of logically related methods on a class that Operations Navigator can call to perform a specific function. A plug-in may implement one or more COM interfaces, depending on the type of function that the developer intends to provide. For example, when a user right-clicks an object in the tree hierarchy, Operations Navigator constructs a context menu for the object and displays the menu on the screen. The Navigator obtains the menu items by calling each plug-in that has registered its desire to supply context menu items for the selected object type. The plug-ins pass their menu items to the Navigator when it calls their implementation of the `QueryContextMenu` method on the `IContextMenu` interface.

### Interface Method Description

<table>
<thead>
<tr>
<th>Interface Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IContextMenu</td>
<td></td>
</tr>
<tr>
<td><code>QueryContextMenu</code></td>
<td>Supplies context menu items when a user right-clicks on an object.</td>
</tr>
<tr>
<td><code>GetCommandString</code></td>
<td>Supplies help text for context menu items and, based on the state of the object, also indicates whether the item should be enabled or grayed.</td>
</tr>
<tr>
<td><code>InvokeCommand</code></td>
<td>Displays the appropriate dialog and performs the requested action. It's called when the user clicks on a given menu item.</td>
</tr>
<tr>
<td>IPropSheetExt</td>
<td></td>
</tr>
<tr>
<td><code>AddPages</code></td>
<td>Creates the property page or pages being added by using standard Windows APIs. It then adds the pages by calling a function that was passed to it as a parameter.</td>
</tr>
<tr>
<td>IDropTarget</td>
<td></td>
</tr>
<tr>
<td><code>DragEnter</code></td>
<td>Active when the user drags an object over the drop area.</td>
</tr>
<tr>
<td><code>DragLeave</code></td>
<td>Active when the user drags an object out of the drop area.</td>
</tr>
<tr>
<td><code>DragOver</code></td>
<td>Active while the user is over the drop area.</td>
</tr>
<tr>
<td><code>Drop</code></td>
<td>Active when the user drops the object.</td>
</tr>
<tr>
<td>IPersistFile</td>
<td></td>
</tr>
<tr>
<td><code>Load</code></td>
<td><em><strong>Are these right? I had to take some guesses here, and couldn't figure anything out for the Load method at all.</strong></em></td>
</tr>
</tbody>
</table>

### IA4 interfaces

In addition to Microsoft's COM interfaces, IBM supplies the IA4HierarchyFolder and IA4PropSheetNotify interfaces.
IA4PropSheetNotify, notifies third-party property pages when the main dialog closes. It also defines methods that communicate information to the plug-in, for example whether the iSeries user whose properties are being displayed already exists or is being defined, and whether changes should be saved or discarded.

IA4HierarchyFolder allows a plug-in to add new folders to the Operations Navigator hierarchy. The purpose of this interface is to supply the data used to populate the contents of a new folder that your plug-in added to the Navigator hierarchy. It also defines methods for specifying list view columns and their headings, and for defining a custom toolbar that is associated with a folder.

See the following topics for more information:
- IA4HierarchyFolder Interface
- IA4HierarchyFolder interface specifications listing
- IA4PropSheetNotify interface
- IA4PropSheetNotify interface specifications listing

**Description of IA4HierarchyFolder Interface**

The IA4HierarchyFolder interface describes a set of functions that the independent software vendor will implement. IA4HierarchyFolder is a component object model (COM) interface that IBM defined for the purpose of allowing third parties to add new folders and objects to the Operations Navigator hierarchy. For a description of the Microsoft COM, see the Microsoft Web site.

The Operations Navigator program calls the methods on the IA4HierarchyFolder interface whenever it needs to communicate with the third-party plug-in. The primary purpose of the interface is to supply the Navigator with list data that will be used when displaying the contents of a folder defined by the plug-in. The methods on the interface allow the Navigator to bind to a particular third-party folder and enumerate its contents. There are methods for returning the number of columns in the details view and their associated headings. Additional methods exist that supply the specifications for a custom toolbar to be associated with the folder.

The interface implementation is typically compiled and linked into an ActiveX server Dynamic Link Library (DLL). The Navigator learns about the existence of the new DLL by means of entries in the Windows registry. These entries specify the location of the DLL on the user’s personal computer and the "junction point" in the object hierarchy where the new folder or folders are to be inserted. The Navigator the loads the DLL at the appropriate time and calls methods on the IA4HierarchyFolder interface as needed.

The header file CWBA4HYF.H contains declarations of the interface prototype and associated data structures and return

**IA4HierarchyFolder interface specifications listing**

A data entity (or “item identifier”) identifies all folders and objects in the Windows namespace. Item identifiers are like filenames in a hierarchical file system. The Windows namespace is, in fact, a hierarchical namespace with its root at the Desktop.

An item identifier consists of a two-byte count field that is followed by a binary data structure of variable length (see the SHITEMID structure in the Microsoft header file SHLOBJ.H). This item identifier uniquely describes an object relative to the parent folder of the object.

The Operations Navigator uses item identifiers that adhere to the following given structure that must be returned by IA4HierarchyFolder::ItemAt.

```xml
<cb><item name="x01"><item type="x02"><item index>
```
where

<cb> is the size in bytes of the item identifier, including the count field itself

=item name= is the translated name of the object, suitable for displaying to the user

=item type= is a unique language-independent string that identifies the object type. It must be at least four characters in length.

=item index= is the zero-based index that identifies the position of the object within the list of parent folder objects.

Link to any of the following IA4HierarchyFolder specifications:

IA4HierarchyFolder::Activate

IA4HierarchyFolder::BindToList

IA4HierarchyFolder::Deactivate

IA4HierarchyFolder::DisplayErrorMessage

IA4HierarchyFolder::GetAttributesOf

IA4HierarchyFolder::GetColumnDataItem

IA4HierarchyFolder::GetColumnInfo

IA4HierarchyFolder::GetIconIndexOf

IA4HierarchyFolder::GetItemCount

IA4HierarchyFolder::GetToolBarInfo

IA4HierarchyFolder::GetListObject

IA4HierarchyFolder::ItemAt

IA4HierarchyFolder::ProcessTerminating
IA4HierarchyFolder::Refresh

IA4CheckIfSupported

Purpose

Called during the scan operation that is performed by Operations Navigator. This API allows the plug-in to provide an indication of whether its function should be surfaced for the specified iSeries system.

Syntax

```c
extern "C" __declspec(dllexport) 
BOOL CALLBACK IA4CheckIfSupported(
    const char * lpszAS400Name,
    const char * lpszVRM,
    const char * lpszInstalledProducts
);
```

Parameters

**lpszAS400Name**

The name of the iSeries server for which the function is to be surfaced.

**lpszVRM**

The Version/Release/Modification level of the specified iSeries server, expressed as a string of the form vvrrmm. For example, OS/400 Version 4 Release 2 would be "040200."

**lpszInstalledProducts**

A large string that contains the product IDs of every product installed on the specified iSeries server. The character "/x01" precedes each product ID in the string.

Return Codes

The function should return TRUE if the plug-in will surface its function for this iSeries server, and FALSE if the plug-in will not display the function.

Comments

Activation of this function occurs only when the ServerEntryPoint field in the primary registry key for the plug-in contains the name of the DLL in which the function resides. During development, a fully-qualified pathname may be specified. When the plug-in is in production, only the name of the DLL should be specified in the registry. The DLL will be loaded from the <VENDOR>,<COMPONENT> subdirectory of the CLIENT_ACCESS/PLUGINS. IA4CheckIfSupported is called on a data thread, not the main UI thread. No user interface tasks should be attempted.

IA4HierarchyFolder::Activate

Purpose

Places the IA4HierarchyFolder instance in an activated state. This function also performs any processing that is needed to prepare a folder for enumeration, including calling the iSeries server to prime the cache of folder objects on the client. The function is called from a data thread so that long running operations will not degrade the performance of the user interface. This is a required member function.

Syntax

```c
HRESULT STDMETHODCALLTYPE Activate();
```
Return Codes

Returns NOERROR if successful or E_FAIL if unable to obtain the contents of the folder.

Comments

The Operations Navigator calls this function the first time a user selects or expands a folder. It is called again, after a call to Close, when the user has requested a refresh of the folder contents.

The function may be called at other times whenever a pointer to the folder interface needs to be reestablished. An example would be when the user selects a folder a second time, after having selected another folder. The function should simply return TRUE if the associated processing has already been performed.

For extremely large lists, you may choose to return from Activate before the list is completely constructed, after having first created a worker thread to continue building the list. If this is the case, make sure that your implementation of GetListSize returns the correct indication of whether the list is completely constructed.

IA4HierarchyFolder::BindToList

Purpose

Returns an instance of IA4HierarchyFolder that corresponds to a particular folder in the Operations Navigator hierarchy. This is a required member function.

Syntax

HRESULT STDMETHODCALLTYPE BindToList(
  HWND hwnd,
  LPCITEMIDLIST pidl,
  REFIID riid,
  LPVOID* ppvOut
);

Parameters

hwnd
Handle of the view window which will display the list (may be either a tree or list control). A component should use this handle to determine if a list of objects for this view is already cached on the client.

pidl
Pointer to an ITEMIDLIST (item identifier list) structure that uniquely identifies the folder to be enumerated.

riid
Identifier of the interface to return. This parameter points to the IID_IA4HierarchyFolder interface identifier.

ppvOut
Address that receives the interface pointer. If an error occurs, a NULL pointer should be returned at this address.

Return Codes

Returns NOERROR if successful or E_FAIL if a general error occurred.

Comments
If an instance of IA4HierarchyFolder already exists for the specified folder, then this member function should return the cached instance instead of instantiating and initializing a separate instance. However, if the window handle associated with the cached object is not the same as the value specified on the hwnd parameter, then a new instance should be created.

The function should initialize implementation class member variables from the parameters supplied.

**IA4HierarchyFolder::DisplayErrorMessage**

**Purpose**

Called to display an error message to the end user whenever Activate returns an error. This is a required member function.

**Syntax**

```cpp
HRESULT STDMETHODCALLTYPE DisplayErrorMessage();
```

**Return Codes**

Returns NOERROR if successful or an E_FAIL if there is no message to display.

**Comments**

**IA4HierarchyFolder::GetAttributesOf**

**Purpose**

Returns the attributes of a particular folder in the Operations Navigator hierarchy. The attribute indicators are the same as those defined for the Microsoft interface method IShellFolder::GetAttributesOf. This is a required member function.

**Syntax**

```cpp
HRESULT STDMETHODCALLTYPE GetAttributesOf(
    LPCITEMIDLIST pidl,
    ULONG* ulfInOut
);
```

**Parameters**

- **pidl**  
  Pointer to an ITEMIDLIST (item identifier list) structure that uniquely identifies the object whose attributes are to be retrieved.

- **ulfInOut**  
  The returned object attributes. On input, this parameter will be set to indicate which object attributes to retrieve.

**Return Codes**

Returns NOERROR if successful or E_FAIL if unable to locate the object attributes.

**Comments**

Refer to the Windows include file shlobj.h for constants that define the bit flags.

This function is called repeatedly by the Operations Navigator when populating a tree or list view. Long running operations should
IA4HierarchyFolder::GetColumnDataItem

Purpose

Returns a data field for a folder or object to be displayed in a column in the list view of the Operations Navigator. This is a required member function.

Syntax

```c
HRESULT STDMETHODCALLTYPE GetColumnDataItem(
    LPCITEMIDLIST pidl,
    LPARAM lParam,
    char * lpszColumnData,
    UINT cchMax
);
```

Parameters

- `pidl` Pointer to an ITEMIDLIST (item identifier list) structure that uniquely identifies the object whose column data is to be obtained.
- `lParam` The value that was previously associated with the column for which data is requested by the component (see GetColumnInfo).
- `lpszColumnData` Address of the buffer that will receive the null-terminated data string.
- `cchMax` Size of the buffer that will receive the null-terminated data string.

Return Codes

Returns NOERROR if successful or an E_FAIL if unable to retrieve the column data.

Comments

This function is called repeatedly by the Operations Navigator when populating a list view. Long running operations should

IA4HierarchyFolder::GetColumnInfo

Purpose

Returns a data structure that describes the columns needed to display the contents of a particular folder in a details view. This is an optional member function.

Syntax

```c
HRESULT STDMETHODCALLTYPE GetColumnInfo(
    LPVOID* ppvInfo
);
```

Parameters

- `ppvInfo` The returned data structure. The returned structure should consist of an instance of the A4hyfColumnInfo structure. This structure contains an array of A4hyfColumnItem structures, one for each column in the list view.
Each column item structure supplies the translated string for the column heading, the default width of the column, and an integer value that uniquely identifies the data field that supplies data for the column. Refer to CWBA4HYF.H.

Return Codes

Returns NOERROR if successful or E_NOTIMPL if unable to implement the function.

Comments

The Operations Navigator calls this function after the call to Open has returned, to create the column headings for a details view.

If this function is not implemented, the Navigator will insert two columns: Name and Description. GetColumnDataItem must be capable of returning data for these two fields, which are identified with integer values of 0 and 1, respectively.

Use the Windows IMalloc interface to allocate memory for the returned structures. The Navigator will be responsible for deleting

IA4HierarchyFolder::GetIconIndexOf

Purpose

Returns the index into the component resource DLL that may be used to load the icon for the hierarchy folder. This is a required member function.

Syntax

HRESULT STDMETHODCALLTYPE GetIconIndexOf(
    LPCITEMIDLIST pidl,
    UINT uFlags,
    int* piIndex
);

Parameters

pidl Pointer to an ITEMIDLIST (item identifier list) structure that uniquely identifies the object whose icon index is to be retrieved.

uFlags This parameter may be zero, or it may contain the value GIL_OPENICON, indicating that the icon which should be supplied is an open folder. GIL_OPENICON is defined in the Windows include file SHOBJ.H.

piIndex Pointer to an integer that receives the icon index.

Return Codes

Returns NOERROR if successful or E_FAIL if unable to determine the index.

Comments

Operations Navigator repeatedly calls this function when populating a tree or list view. Long running operations should

IA4HierarchyFolder::GetItemCount

Purpose

Chapter 5. Plug-in developers reference
Returns the total count of objects contained in a particular folder in the Operations Navigator hierarchy. This is a required member function.

Syntax

```
HRESULT STDMETHODCALLTYPE GetItemCount(
    ULONG* pCount
);
```

Parameters

- `pCount` Pointer to a long integer that will receive the count of items in the list.

Return Codes

Returns A4HYF_OK_LISTCOMPLETE if the list is completely built and the total count of items is known. Returns A4HYF_OK_LISTNOTCOMPLETE if the list is still being constructed; in this situation the item count represents the count of items in the partially constructed list. Returns A4HYF_E_LISTDATAERROR if an error is encountered while constructing the list; in this situation the item count represents only the items that are already cached on the client.

Comments

Following a successful return from Activate, Navigator calls this function to obtain the count of objects for the folder that is about to be populated. Following the call to this function, Navigator repeatedly calls ItemAt to obtain the item identifiers for the objects in the folder.

For extremely large lists, you may choose to return from Activate before the entire list has been cached on the client. If this is the case, you should return A4HYF_OK_LISTNOTCOMPLETE from GetItemCount. From that point on, GetItemCount will be called by the Operations Navigator every 10 seconds until A4HYF_OK_LISTCOMPLETE or A4HYF_E_LISTDATAERROR is returned.

IA4HierarchyFolder::GetListObject

Purpose

Given a fully-qualified object name, this function returns a pointer to a cached proxy object created by the plug-in. This is an optional member function.

Syntax

```
HRESULT STDMETHODCALLTYPE GetListObject(
    const char * lpszObjectName,
    LPVOID* ppvObj
);
```

Parameters

- `lpszObjectName` The fully-qualified object name for which a list object will be returned.

- `ppvObj` The returned pointer to an implementation-defined object. The calling routine should cast this pointer to an appropriate object type.

Return Codes
Returns NOERROR if successful or E_NOTIMPL if you choose not to implement the function.

Comments

Calls to this function occur whenever your plug-in code calls cwbUN_GetListObjectFromName or cwbUN_GetListObjectFromPidl to obtain a “proxy” object that was instantiated by the Activate method. The plug-in uses this proxy object to access data on the iSeries server, or to perform actions on the iSeries server. Because IA4HierarchyFolder implementation maintains the cache of proxy objects, the calling program should not attempt to

IA4HierarchyFolder::GetToolBarInfo

Purpose

Returns a structure that describes the custom tool bar that is associated with the specified folder in the Operations Navigator hierarchy. This is a required member function.

Syntax

HRESULT STDMETHODCALLTYPE GetToolBarInfo(
    LPCITEMIDLIST pidl,
    LPVOID* ppvInfo
);

Parameters

pidl Pointer to an ITEMIDLIST (item identifier list) structure that uniquely identifies the object for which tool bar information is to be retrieved.

ppvInfo The returned data structure. An instance of A4hyfToolBarInfo should be returned in this pointer. This structure supplies the count of toolbar buttons for the object, the address of an array of TBBUTTON structures containing the attributes for each button, and the instance handle of the plug-in. Refer to CWBA4HYF.H.

Return Codes

Returns NOERROR if successful or E_NOTIMPL if you choose not to implement the function.

Comments

This function is called each time a user selects a folder or object that belongs to an Operations Navigator plug-in.

Use the Windows IMalloc interface to allocate memory for the returned structure. The Navigator will be responsible for deleting this memory.

If this member function is not implemented, the default Operations Navigator tool bar will be used. This toolbar contains Copy, Paste, Delete, Properties, buttons for the four list views, and Refresh. The Operations Navigator calls the implementation of IContextMenu::GetCommandString (with the GCS_VALIDATE flag set) that is in your product to discover which of the toolbar buttons

IA4HierarchyFolder::ItemAt

Purpose

Returns as SHITEMID (item identifier) structure for the folder object at the specified position in the list of folder contents. This is a required member function.
Syntax

HRESULT STDMETHODCALLTYPE ItemAt(
    ULONG ulIndex,
    LPITEMIDLIST* ppidl
);

Parameters

ulIndex  The zero-based index of the item for which an item identifier is requested.

ppidl    Address of the pointer that will receive the requested item identifier.

Return Codes

Returns NOERROR if successful or E_FAIL if the item is not available. Returns E_OUTOFMEMORY if
insufficient memory was available for the item identifier.

Comments

The Operations Navigator repeatedly calls this function to populate a folder in realtime. Long running
operations should therefore be avoided. Refer to CWBA4HYF.H for the format of Operations Navigator
item identifiers. Use the Windows IMalloc interface to allocate memory for the item identifier.

IA4HierarchyFolder::ProcessTerminating

Purpose

Call to this function occur when the user closes the Operations Navigator window to provide the plug-in
with an opportunity to save persistent data. This is an optional member function.

Syntax

HRESULT STDMETHODCALLTYPE ProcessTerminating();

Return Codes

Returns NOERROR if successful or E_NOTIMPL if you choose not to implement the function. Error returns
are ignored.

Comments

IA4HierarchyFolder::Refresh

Purpose

Destroys any cached folder objects and rebuilds the cache using new data obtained from the iSeries
server. This is a required member function.

Syntax

HRESULT STDMETHODCALLTYPE Refresh();

Return Codes
Returns NOERROR if successful or A4HYF_E_LISTDATAERROR if an error occurred when accessing the objects in the folder.

Comments

Operations Navigator calls this function is called whenever a performing a global refresh of the main Operations Navigator.

Description of IA4PropSheetNotify interface

Like the IA4HierarchyFolder interface, the IA4PropSheetNotify interface describes a set of functions that the independent software vendor will implement. IA4PropSheetNotify is a COM interface IBM defined to allow third parties to add new property pages to any property sheet that the Operations Navigator defines for an iSeries server user.

The Operations Navigator program calls the methods on the IA4PropSheetNotify interface whenever it needs to communicate with the third-party plug-in. The purpose of the interface is to provide notification when the main Properties dialog for an iSeries user is closing. The notification indicates whether any changes that are made by the user should be saved or discarded. The intention is that the interface be added to the same implementation class that is used for IPropSheetExt.

The interface implementation is compiled and linked into the ActiveX server DLL for the plug-in. The Navigator learns of the existence of the new DLL by means of entries in the Windows registry. These entries specify the location of the DLL on the user’s personal computer. The Navigator then loads the DLL at the appropriate time, calling methods on the IA4PropSheetNotify interface as needed.

CWBA4HYF.H contains declarations of the interface prototype and associated data structures and return codes.

IA4PropSheetNotify interface specifications listing

The IA4PropSheetNotify interface supplies notifications to the implementation of IShellPropSheetExt that are needed when adding additional property pages to one of the Users and Groups property sheets. These notifications are necessary because creating and destroying Users and Groups property sheets may occur many times before the user clicks OK on the main Properties dialog. IA4PropSheetNotify informs the IShellPropSheetExt implementation when changes that are made by the user should be saved.

The Operations Navigator learns about an IA4PropSheetNotify implementation by means of the normal registry entries that are defined for Operations Navigator plug-ins. In addition, when a property sheet handler for the Users and Groups component is registered, a special registry value is supported that allows the plug-in to specify to which property sheet it desires to add pages.

Link to any of the following IA4PropSheetNotify interface specifications:
- IA4PropSheetNotify::InformUserState
- IA4PropSheetNotify::ApplyChanges
- IA4PropSheetNotify::GetErrorMessage

IA4PropSheetNotify::ApplyChanges

Purpose

Called to inform the implementation that data that belongs to the user should now be saved.

Syntax
HRESULT STDMETHODCALLTYPE ApplyChanges(
    const char * pszNewUserName
);

Parameters

pszNewUserName
    Supplies the name of the new iSeries user if creating the user for the first time, for example, if InformUserState specifies a value other than IUS_USEREXISTS.

Return Codes

Returns NOERROR if successful or E_FAIL if a general error occurred.

Comments

IA4PropSheetNotify::GetErrorMessage
Purpose

Called when errors are returned on ApplyChanges to retrieve the implementation’s error message text.

Syntax

HRESULT STDMETHODCALLTYPE GetErrorMessage(
    char * pszErrMsg,
    UINT cchMax
);

Parameters

pszErrMsg
    Address of the buffer that will receive the null-terminated error message.

cchMax
    Size of the buffer that will receive the null-terminated error message.

Return Codes

Returns NOERROR if successful or E_FAIL if unable to retrieve the message text or if message text was too large to fit in the buffer.

Comments

IA4PropSheetNotify::InformUserState
Purpose

Called immediately following creation of the IShellPropSheetExt instance to inform the implementation whether this user already exists on the iSeries server or is being created for the first time.

Syntax

HRESULT STDMETHODCALLTYPE InformUserState(
    UINT wUserState
);

Parameters

wUserState
wUserState

The current state of the iSeries user. Supplies one of three mutually exclusive values:

**IUS_NEWUSER**
Creating a user based on attributes that are supplied by the Operations Navigator user.

**IUS_NEWUSERBASEDON**
Creating a user based on the attributes of an existing iSeries user.

**IUS_USEREXISTS**
The user already exists on the iSeries.

Return Codes

Returns NOERROR if successful or E_FAIL if a general error occurred.

Comments

Operations Navigator API listing

Operations Navigator APIs help plug-in developers obtain and manage certain types of global information. The following Operations Navigator APIs are listed alphabetically, and are grouped by function:

<table>
<thead>
<tr>
<th>Function</th>
<th>Operations Navigator APIs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System values:</strong> This API allows the plug-in developer to obtain the current value of an iSeries system value.</td>
<td>cwbUN_GetSystemValue [on page 76]</td>
</tr>
<tr>
<td><strong>System handles:</strong> These APIs allow the plug-in developer to obtain and to release the current value of an iSeries system object handle that contains connection properties including the secure sockets layer (SSL) settings to be used for the specified iSeries system.</td>
<td>cwbUN_GetSystemHandle</td>
</tr>
<tr>
<td>cwbUN_ReleaseSystemHandle</td>
<td></td>
</tr>
<tr>
<td><strong>User input validation:</strong> These APIs allow the plug-in developer to check whether the current user has authority to a particular iSeries object. The APIs also allow the developer to determine if the user has one or more special authorities.</td>
<td>cwbUN_CheckObjectAuthority</td>
</tr>
<tr>
<td>cwbUN_CheckSpecialAuthority</td>
<td></td>
</tr>
<tr>
<td><strong>User authority checking:</strong> This API allows the plug-in developer to check whether certain types of user-supplied strings are valid before transmitting them to the iSeries server.</td>
<td>cwbUN_CheckAS400Name [on page 46]</td>
</tr>
<tr>
<td><strong>User profile attributes:</strong> This API allows the plug-in developer to obtain the value of any of the user profile attributes for the current Operations Navigator user.</td>
<td>cwbUN_GetUserAttribute</td>
</tr>
</tbody>
</table>
Function

Data management: Objects that the user has selected are identified to the third-party plug-in by two data entities, the item identifier list, and the object name. Data management APIs provide the plug-in developer with a means of extracting information from these structures.

Operations Navigator APIs

cwbUN_ConvertPidlToString

cwbUN_GetDisplayNameFromItemId

cwbUN_GetDisplayNameFromName

cwbUN_GetDisplayNameFromPathFromName

cwbUN_GetIndexFromItemId

cwbUN_GetIndexFromName

cwbUN_GetIndexFromPidl

cwbUN_GetListObject

cwbUN_GetParentFolderPathFromName

cwbUN_GetParentFolderPathFromItem

cwbUN_GetParentFolderNameFromName

cwbUN_GetSystemNameFromName

cwbUN_GetSystemNameFromPidl

cwbUN_GetTypeFromItemId

cwbUN_GetTypeFromName

cwbUN_GetTypeFromPidl
Function

Refresh the Operations Navigator window: Following the completion of an operation on behalf of the user, these APIs enable execution of a request by the plug-in to refresh the tree and list views or to place a message in the Navigator status bar.

- cwbUN_RefreshAll
- cwbUN_RefreshList
- cwbUN_RefreshListItems
- cwbUN_UpdateStatusBar

ODBC connections: These APIs allow the plug-in developer to reuse and end the handle for an ODBC connection that already has been obtained by the Database component of the Operations Navigator.

- cwbUN_GetODBCConnection
- cwbUN_EndODBCConnections

Access Operations Navigator icons: These APIs allow the plug-in developer to access the icon image lists for objects that appear in the Navigator object hierarchy.

- cwbUN_GetIconIndex
- cwbUN_GetSharedImageList

Application Administration: These APIs allow the plug-in developer to programmatically determine whether a user is denied or allowed use of an Administrable function. An Administrable function is any function whose use can be controlled through the Application Administration sub component of Operations Navigator.

- cwbUN_GetAdminValue
- cwbUN_GetAdminValueEx
- cwbUN_GetAdminCacheState
- cwbUN_GetAdminCacheStateEx

Install: This API allows the plug-in developer to determine if an Operations Navigator sun component is installed.

- cwbUN_IsSubcomponentInstalled
Function

**Directory Services:** These APIs provide information about the Directory Services (LDAP) server on an iSeries computer, and functions to connect to the server. The connection functions enable you to connect to a server using information (distinguished names, password, etc.) cached by the Client Access Express. The connection functions use the LDAP client shipped with Client Access (LDAP.LIB and LDAP.DLL) and therefore require that your application use that client.

Functions that use strings are available in ANSI and Unicode versions.

Functions that return distinguished names and other strings for use with LDAP client APIs also are provided in a UTF-8 version for use with LDAP version 3 servers.

See FORM='TEXTONLY' for more information.

### Operations Navigator APIs

- `cwbUN_OpenLocalLdapServer`
- `cwbUN_FreeLocalLdapServer`
- `cwbUN_GetLdapSvrPort` [on page 67]
- `cwbUN_GetLdapSvrSuffixCount`
- `cwbUN_GetLdapSvrSuffixName`
- `cwbUN_OpenLdapPublishing`
- `cwbUN_FreeLdapPublishing`
- `cwbUN_GetLdapPublishCount`
- `cwbUN_GetLdapPublishType`
- `cwbUN_GetLdapPublishServer`
- `cwbUN_GetLdapPublishPort`
- `cwbUN_GetLdapPublishParentDn`
- `cwbUN_OpenLdapBindInfo`
- `cwbUN_FreeLdapBindInfo`
- `cwbUN_GetLdapServerBindDn`
- `cwbUN_BindToLdapServerOnAs400`
- `cwbUN_BindToLdapServer`
- `cwbUN_NullBindToLdapServerOnAs400`
- `cwbUN_NullBindToLdapServer`
cwbUN_BindToLdapServer

Purpose

Binds to the specified server using information cached by Client Access including:

- Distinguished name
- Password
- SSL configuration

If any required information is not available, a dialog is displayed to collect the data. If an error occurs, a message is displayed to the user.

The connection is established through the Client Access LDAP client (LDAP.LIB and LDAP.DLL), and only can be used with that client.

This API will attempt to establish bind using LDAP version 3, and then LDAP version 2. Use the ldap_get_option LDAP client API to determine what version the bind used if you need to use LDAP version 3 client APIs with the connection.

Syntax

```c
int cwbUN_BindToLdapServerA
( LPCSTR server,
  int port,
  int sslPort,
  cwbUN_LdapBindOption options,
  int version,
  HWND hwnd,
  void **LPldap
);

int cwbUN_BindToLdapServerW
( LPCWSTR server,
  int port,
  int sslPort,
  cwbUN_LdapBindOption options,
  int version,
  HWND hwnd,
  void **LPldap
);
```

Parameters

- **LPCSTR system** - input
  
  NULL-terminated iSeries system name or dotted IP address.

- **int port** - input
  
  The port to be used for standard connections. Specify 0 to force an SSL connection.

- **int sslPort** - input
  
  The port to be used for SSL connections. Specify 0 if SSL connections are not supported by the server.

- **cwbUN_LdapBindOption options** - input
  
  Specifies options controlling the display of error messages and the "Connect to Directory Server" dialog. Possible values are:
CWBUN_LDAP_BINDOPT_NO_DLG
Do not show the "Connect To Directory Server" dialog and do not display any error messages.

CWBUN_LDAP_BINDOPT_DLG_ON_ERR
Show the "Connect To Directory Server" dialog and error messages. The user has the opportunity to change bind information if an error occurs.

CWBUN_LDAP_BINDOPT_SHOW_DLG
Show the "Connect to Directory Server" dialog before attempting to connect to the server. The user has the opportunity to change bind information before connecting to the server.

int version - input
Specifies the LDAP API version to use for binding to the server. Possible values are:

LDAP_VERSION2 (from ldap.h)
Bind as a LDAP version 2 client.

LDAP_VERSION3 (from ldap.h)
Bind as a LDAP version 3 client. If the server does not support version 3, the bind will fail with a protocol error.

CWBUN_LDAP_VERSION_LATEST
Bind as a LDAP version 3 client if supported by the server. Otherwise, bind as a LDAP version 2 client.

LDAP **LPldap - output
On return, *LPldap points to the LDAP returned by the LDAP bind APIs.

HWND hwnd - input
Window handle to be used for messages. This handle also will be used as the parent window of the "Connect to Directory server" dialog. NULL can be specified.

Return Codes
The following list shows common return values:

CWB_OK
Successful completion.

CWB_INVALID_API_PARAMETER
Invalid parameter specified.

CWB_INVALID_POINTER
A NULL pointer was specified.

CWBUN_LDAP_BIND_FAILED
A connection could not be established to the server.

Usage
cwbUN_BindToLdapServerOnAs400
Purpose
Binds to the specified server using information cached by Client Access including:
• Distinguished name
• Password
• SSL configuration
If any required information is not available, a dialog is displayed to collect the data. If an error occurs, a message is displayed to the user.

The connection is established through the Client Access LDAP client (LDAP.LIB and LDAP.DLL), and only can be used with that client.

This API will attempt to establish bind using LDAP version 3, and then LDAP version 2. Use the `ldap_get_option` LDAP client API to determine what version the bind used if you need to use LDAP version 3 client APIs with the connection.

### Syntax

```c
int cwbUN_BindToLdapServerOnAs400A
    ( LPCSTR system,
      cwbUN_LdapBindOption options,
      int version,
      HWND hwnd,
      void **LPldap
    );

int cwbUN_BindToLdapServerOnAs400W
    ( LPCWSTR system,
      cwbUN_LdapBindOption options,
      int version,
      HWND hwnd,
      void **LPldap
    );
```

### Parameters

**LPCSTR system** - input

NULL terminated iSeries system name.

**cwbUN_LdapBindOption options** - input

Specifies options controlling the display of error messages and the "Connect to Directory Server" dialog. Possible values are:

- **CWBUN_LDAP_BINDOPT_NO_DLG**
  Do not show the "Connect To Directory Server" dialog and do not display any error messages.

- **CWBUN_LDAP_BINDOPT_DLG_ON_ERR**
  Show the "Connect To Directory Server" dialog and error messages. The user has the opportunity to change bind information if an error occurs.

- **CWBUN_LDAP_BINDOPT_SHOW_DLG**
  Show the "Connect to Directory Server" dialog before attempting to connect to the server. The user has the opportunity to change bind information before connecting to the server.

**int version** - input

Specifies the LDAP API version to use for binding to the server. Possible values are:

- **LDAP_VERSION2** (from ldap.h)
  Bind as a LDAP version 2 client.

- **LDAP_VERSION3** (from ldap.h)
  Bind as a LDAP version 3 client. If the server does not support version 3, the bind will fail with a protocol error.
**CWBUN_LDAP_VERSION_LATEST**

Bind as a LDAP version 3 client if supported by the server. Otherwise, bind as a LDAP version 2 client.

**LDAP **LPldap - output**

On return, *LPldap points to the LDAP* returned by the LDAP bind APIs.

**HWND hwnd - input**

Window handle to be used for messages. This handle also will be used as the parent window of the "Connect to Directory server" dialog. NULL can be specified.

**Return Codes**

The following list shows common return values:

**CWB_OK**

Successful completion.

**CWB_INVALID_API_PARAMETER**

Invalid parameter specified.

**CWB_INVALID_POINTER**

A NULL pointer was specified.

**CWBUN_LDAP_NOT_AVAIL**

Directory services is not installed or the server has not been configured.

**CWBUN_LDAP_BIND_FAILED**

A connection could not be established to the server.

**Usage**

**cwbUN_CheckAS400Name**

**Purpose**

Returns an indication of whether a specified string is a valid name parameter on the iSeries server.

**Syntax**

```c
CWBAPI unsigned int WINAPI cwbUN_CheckAS400Name(
    const char * szAS400Name,
    const char * szSystemName,
    USHORT usTypeId
);
```

**Parameters**

**const char * szAS400Name - input**

The iSeries name whose validity is to be checked.

**const char * szSystemName - input**

The name of the iSeries system on which to perform the check.

**USHORT usTypeId - input**

A numeric value that indicates how the input string should be interpreted: as a long object name, a short object name, a communications name, or a string (type constants are defined above).

**Return Codes**

46
The following list shows common return values:

**CWB_OK**
Successful completion.

**CWBUN_NAME_TOO_LONG**
Name is too long.

**CWBUN_NAME_NULLSTRING**
String is empty - no characters at all.

**CWBUN_NAME_INVALIDCHAR**
Character not valid.

**CWBUN_NAME_STRINGTOOLONG**
String is too long.

**CWBUN_NAME_MISSINGENDQUOTE**
End quote is missing.

**CWBUN_NAME_INVALIDQUOTECHAR**
Character not valid for quote string.

**CWBUN_NAME_ONLYBLANKS**
Found a string of only blanks.

**CWBUN_NAME_STRINGTOOSHORT**
String is too short.

**CWBUN_NAME_TOOLONGFORIBM**
String is OK, but too long for IBM command.

**CWBUN_NAME_INVALIDFIRSTCHAR**
The first character is not valid.

Usage

cwbUN_CheckObjectAuthority

Purpose

Returns an indication of whether the Operations Navigator user has authority to a particular object on the iSeries server.

Syntax

CWBAPI unsigned int WINAPI cwbUN_CheckObjectAuthority(
    const char * szObjectPath,
    const char * szObjectType,
    const char * szAuthorityType,
    const char * szSystemName
);

Parameters

**const char * szObjectPath - input**
The iSeries object path for which authority is to be checked.

**const char * szObjectType - input**
The iSeries object type for the object for which authority is to be checked, for example *DTAQ.

**const char * szAuthorityType - input**
The iSeries object authority to be checked. Possible values are:
EXCLUDE

CHANGE

USE

AUTLMGT

OBJALTER

OBJOPR

OBJMGT

OBJEXIST

OBJREF

READ

ADD

UPD

DLT

EXECUTE

If more than one authority is to be checked, the authorities should be concatenated (for example, "*OBJMGT*OBJEXIST"). Up to eleven authority types may be specified on a single call. The function will return CWB_OK only if the user has all of the specified authorities to the object.

const char * szSystemName - input
   The name of the iSeries system on which to perform the check.

Return Codes

The following list shows common return values:

CWB_OK
   The user has the specified authority to the object.
CWBUN_USER_NOT_AUTHORIZED
The user does not have the specified authority.

CWBUN_OBJECT_NOT_FOUND
The specified object could not be checked.

CWBUN_INTERNAL_ERROR
Object authority could not be checked.

Usage
If "EXCLUDE is specified as an authority, no other authority types may be specified. "AUTLMGT is valid only if 'szObjectType' is

cwbUN_CheckSpecialAuthority

Purpose
Returns an indication of whether the Operations Navigator user has a particular special authority on the iSeries server.

Syntax
CWBAPI unsigned int WINAPI cwbUN_CheckSpecialAuthority(
    const char * szSpecialAuthority,
    const char * szSystemName
);

Parameters
const char * szSpecialAuthority - input
The iSeries server special authority to be checked. Possible values are:

*ALLOBJ

*AUDIT

*IOSYSCFG

*JOBCTL

*SAVSYS

*SECADM

*SERVICE

*SPLCTL
const char * szSystemName - input

The name of the iSeries system on which to perform the check.

Return Codes

The following list shows common return values:

CWB_OK

The user has the specified special authority.

CWBUN_USER_NOT_AUTHORIZED

The user does not have the specified authority.

CWBUN_INTERNAL_ERROR

Special authority could not be checked.

Usage

cwbUN_ConvertPidlToString

Purpose

Converts an Operations Navigator item identifier list to a fully-qualified object name.

Syntax

CWBAPI unsigned int WINAPI cwbUN_ConvertPidlToString(

LPCITEMIDLIST pidl,

char * szObjectName,

UINT cchMax

);

Parameters

LPCITEMIDLIST pidl - input

Pointer to the ITEMIDLIST (item identifier list) structure that is to be converted.

char * szObjectName - output

Address of the buffer that will receive the null-terminated object name.

UINT cchMax - input

Size of the buffer that will receive the null-terminated object name.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWBUN_FORMAT_NOT_VALID

The specified item identifier list is not valid.

WB_BUFFER_OVERFLOW

The buffer is too small to contain the returned string.

Usage
**cwbUN_EndODBCCConnections**

**Purpose**

Ends all ODBC connections previously opened by `cwbUN_GetODBCCConnection`.

**Syntax**

```c
CWBAPI unsigned int WINAPI cwbUN_EndODBCCConnections(
);
```

**Parameters**

None

**Return Codes**

The following list shows common return values:

- **CWB_OK**
  - Successful completion.
- **CWB_INVALID_API_HANDLE**
  - Handle was not created by `cwbUN_OpenLdapBindInfo()`.

**Usage**

It is important to remember that the `EndODBCCConnections` function only closes connections that were opened using the `GetODBCCConnection` function. The `EndODBCCConnections` function is unaware of ODBC connections opened directly or by using other interfaces.

Also ensure that the destructor for the folder of your application extension invokes the `EndODBCCConnections` if any code in your extension uses `GetODBCCConnection`.

See also `cwbUN_GetODBCCConnection`.

**cwbUN_FreeLdapBindInfo**

**Purpose**

Frees resources associated with the input handle.

**Syntax**

```c
int cwbUN_FreeLdapBindInfo
    ( cwbUN_ladBindInfoHandle handle
    );
```

**Parameters**

- **cwbUN_ladBindInfoHandle handle - input**
  - The handle for which resources should be freed.

**Return Codes**

The following list shows common return values:

- **CWB_OK**
  - Successful completion.
CWB_INVALID_API_HANDLE
Handle was not created by cwbUN_OpenLdapBindInfo().

Usage
The handle is obtained by a call to cwbUN_OpenLdapBindInfoW().

cwbUN_FreeLdapPublishing
Purpose
Frees resources associated with the input handle.

Syntax

```c
int cwbUN_FreeLdapPublishing( cwbUN_ldapPubHandle handle );
```

Parameters

cwbUN_ldapPubHandle handle - input
The handle for which resources should be freed.

Return Codes
The following list shows common return values:

CWB_OK
Successful completion.

CWB_INVALID_API_HANDLE
Handle was not created by cwbUN_OpenLdapPublishing().

Usage
The handle is obtained by a call to cwbUN_OpenLdapPublishing().

cwbUN_FreeLocalLdapServer
Purpose
Frees resources associated with the input handle.

Syntax

```c
int cwbUN_FreeLocalLdapServer( cwbUN_ldapSvrHandle handle );
```

Parameters

cwbUN_ldapSvrHandle handle - input
The handle for which resources should be freed.

Return Codes
The following list shows common return values:
CWB_OK
  Successful completion.

CWB_INVALID_API_HANDLE
  handle was not created by cwbUN_OpenLocalLdapServer()

Usage

The handle is obtained by a call to cwbUN_OpenLocalLdapServer.

**cwbUN_GetAdminCacheState**

**Purpose**

This API indicates whether the next invocation of the cwbUN_GetAdminValue API will be long running. The cwbUN_GetAdminValue API caches data on the PC. If the cache is not current, cwbUN_GetAdminValue may present a sign-on prompt, or perform other processing, in order to update its cache.

**Syntax**

```c
CWBAPI unsigned int WINAPI cwbUN_GetAdminCacheState(
    const char * szSystemName,
    cwbUN_State& adminState);
```

**Parameters**

**const char * szSystemName**
  The name of the iSeries system on which to perform the check.

**cwbUN_State& adminState**
  Indicates if the next invocation of the cwbUN_GetAdminValue API will be long running or if it will use its internal cache to return without accessing the host iSeries server.

One of three values will be returned:

**cwbUN_logon**
  There is no current user for the specified iSeries server. The cwbUN_GetAdminValue API may present a sign-on prompt.

**cwbUN_refresh**
  cwbUN_GetAdminValue will access the iSeries server to update its internal cache.

**cwbUN_cache**
  cwbUN_GetAdminValue has a current cache and should not be long running.

**Return Codes**

The following list shows common return values:

**CWB_OK**
  The API was successful.

**Usage**

This API can be used by users of cwbUN_GetAdminValue to determine if the next invocation of cwbUN_GetAdminValue will be
cwbUN_GetAdminCacheStateEx

Purpose

This API indicates whether the next invocation of the cwbUN_GetAdminValueEx API will be long running. The cwbUN_GetAdminValueEx API caches data on the PC. If the cache is not current, the cwbUN_GetAdminValueEx API may present a sign-on prompt, or perform other processing, in order to update its cache.

Syntax

```c
CWBAPI unsigned int WINAPI cwbUN_GetAdminCacheStateEx(
   cwbCO_SysHandle* pSysHandle,
   cwbUN_State& adminState);
```

Parameters

cwbCO_SysHandle* pSysHandle - input
A pointer to a system object handle. The system name must be specified in the system object prior to calling this API.

cwbUN_State& adminState
Indicates if the next invocation of the cwbUN_GetAdminValue API will be long running or if it will use its internal cache to return without accessing the host iSeries server.

One of three values will be returned:

cwbUN_logon
There is no current user for the specified iSeries server. The cwbUN_GetAdminValue API may present a sign-on prompt.

cwbUN_refresh
cwbUN_GetAdminValue will access the iSeries server to update its internal cache.

cwbUN_cache
cwbUN_GetAdminValue has a current cache and should not be long running.

Return Codes

The following list shows common return values:

CWB_OK
The API was successful.

Usage

This API can be used by users of cwbUN_GetAdminValueEx to determine if the next invocation of cwbUN_GetAdminValueEx will be

cwbUN_GetAdminValue

Purpose

This API returns an indication of whether the current Operations Navigator user on the specified iSeries server is allowed or denied use of a specific administrable function. An Administrable function is any function whose use can be controlled through the Application Administration subcomponent of Operations Navigator.
For example, the Application Administration subcomponent allows an administrator to control whether a user can access several functions in Operations Navigator. One of these functions is Job Management. The cwbUN_GetAdminValue API can be used to programmatically determine if the current Operations Navigator user can use the Job Management function by specifying the name of the Administrable function that corresponds to Job Management. See the cwbunpla.h header file for a list of Administrable function names that are supported in Operations Navigator.

Syntax

```c
CWBAPI unsigned int WINAPI cwbUN_GetAdminValue(
    const char * szSystemName,
    char* adminFunction,
    cwbUN_Usage& usageValue);
```

Parameters

**const char * szSystemName**
- The name of the iSeries system on which to perform the check.

**char* adminFunction**
- A pointer to an ASCII string that contains the name of the Administrable function. The string must be null terminated and has a maximum length of 30 bytes + 1 byte for the NULL terminator. See cwbunpla.h for a list of supported input values.

**cwbUN_Usage & usageValue**
- This value is only valid if the return code of CWB_OK is returned. One of two values will be returned:
  - cwbUN_granted — User is allowed use of the function.
  - cwbUN_denied — user is denied use of the function.

Return Codes

The following list shows common return values:

**CWB_OK**
- The API was successful.

**CWBSY_USER_CANCELLED**
- The user cancelled the user ID and password prompt presented by the API.

Usage

This API determines if the current Operations Navigator user for the specified iSeries server is allowed to use the specified function. If no user is currently signed on to the specified iSeries server, the API will sign the user on, possibly displaying a user ID and password prompt.

This API can only be used to check Administrable functions that are in the Operations Navigator or the Client Applications

**cwbUN_GetAdminValueEx**

Purpose

This API returns an indication of whether the current user on the specified iSeries server is allowed or denied use of a specific administrable function. An Administrable function is any function whose use can be controlled through the Application Administration subcomponent of Operations Navigator.
Note: Operations Navigator plug-ins should use the cwbUN_GetAdminValue API instead of cwbUN_GetAdminValueEx.

For example, the Application Administration subcomponent allows an administrator to control whether a user can access several functions in Operations Navigator. One of these functions is 'Job Management'. The cwbUN_GetAdminValueEx API can be used to programmatically determine if the current user can use the Job Management function by specifying the name of the Administrable function that corresponds to Job Management. See the CWBUNPLA.H header file for a list of Administrable function names that are supported in Operations Navigator.

This API provides the same function as cwbUN_GetAdminValue, except that it is designed to accept a system object handle instead of a system name.

Syntax

```c
CWBAPI unsigned int WINAPI cwbUN_GetAdminValueEx(
    cwbCO_SysHandle* pSysHandle,
    char* adminFunction,
    cwbUN_Usage& usageValue);
```

Parameters

cwbCO_SysHandle* pSysHandle
A pointer to a system object handle. The system name must be specified in the system object prior to calling this API. The cwbUN_GetAdminValueEx API’s behavior is based on whether the system object has obtained a signon to the iSeries server:

Not Signed On->
cwbUN_GetAdminValueEx will signon to the iSeries server. The latest Application Administration settings for the user will be downloaded from the iSeries server if they are not already cached on the client PC.

Signed On->
If the system object was signed on to the iSeries server specifying that the iSeries userID and password should be validated (Validate Mode), then the cwbUN_GetAdminValueEx API will be using a snapshot of Application Administration settings that were accurate at the time the signon was completed. If the signon was done without validating the userID and password, then it is possible that cwbUN_GetAdminValueEx will use a copy of the Application Administration settings that may be as much as 24 hours old.

cchar* adminFunction
A pointer to an ASCII string that contains the name of the Administrable function. The string must be null terminated and has a maximum length of 30 bytes + 1 byte for the NULL terminator. See CWBUNPLA.H for a list of supported input values.

cwbUN_Usage& usageValue
This value is only valid if the return code of CWB_OK is returned. One of two values will be returned:

cwbUN_granted
User is allowed use of the function.

cwbUN_denied
User is denied use of the function.
Return Codes

The following list shows common return values:

**CWB_OK**

The API was successful.

**CWBSY_USER_CANCELLED**

The user cancelled the user ID and password prompt presented by the API.

Usage

This API determines if the current iSeries user (as defined by the input system object) is allowed to use the specified function. If no user is currently signed on to the specified iSeries server, the API will sign the user on, possibly displaying a user ID and password prompt.

This API can only be used to check Administrable functions that are in the Operations Navigator or the Client Applications

cwbUN_GetDisplayNameFromItemId

Purpose

Extracts the item name field from a Unity item identifier.

Syntax

```c
CWBAPI unsigned int WINAPI cwbUN_GetDisplayNameFromItemId(
    const char * szItemId,
    char * szItemName,
    UINT cchMax
);
```

Parameters

- **const char * szItemId - input**
  
  The Unity item identifier from which the item name will be extracted.

- **char * szItemName - output**
  
  Address of the buffer that will receive the null-terminated item name.

- **UINT cchMax - input**
  
  Size of the buffer that will receive the null-terminated item name.

Return Codes

The following list shows common return values:

**CWB_OK**

Successful completion.

**CWBUSN_FORMAT_NOT_VALID**

Specified item identifier not valid.

**CWB_BUFFER_OVERFLOW**

The buffer is too small to contain the returned string.
cwbUN_GetDisplayNameFromName

Purpose

Extracts the item name field from a fully-qualified Unity object name.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetDisplayNameFromName(
    const char * szObjectName,
    char * szItemName,
    UINT cchMax
);

Parameters

const char * szObjectName - input
    The Unity object name from which the item name will be extracted.

char * szItemName - output
    Address of the buffer that will receive the null-terminated item name.

UINT cchMax - input
    Size of the buffer that will receive the null-terminated item name.

Return Codes

The following list shows common return values:

CWB_OK
    Successful completion.

CWBUN_FORMAT_NOT_VALID
    Specified object name is not valid.

CWB_BUFFER_OVERFLOW
    The buffer is too small to contain the returned string.

Usage

cwbUN_GetDisplayPathFromName

Purpose

Converts a fully-qualified Unity object name to a fully-qualified pathname suitable for displaying to the user.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetDisplayPathFromName(
    const char * szObjectName,
    char * szPathName,
    UINT cchMax
);

Parameters

const char * szObjectName - input
    The Unity object name from which the pathname will be derived.

char * szPathName - output
    Address of the buffer that will receive the null-terminated pathname.
UINT cchMax - input
    Size of the buffer that will receive the null-terminated pathname.

Return Codes

The following list shows common return values:

CWB_OK
    Successful completion.

CWBUN_FORMAT_NOT_VALID
    Specified object name is not valid.

CWB_BUFFER_OVERFLOW
    The buffer is too small to contain the returned string.

Usage

cwbUN_GetIconIndex

Purpose

Get the index in the image list of the specified icon.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetIconIndex(
    LPCITEMIDLIST pidl,
    UINT uFlags,
    int* piIndex
);

Parameters

LPCITEMIDLIST pidl - input
    Pointer to the ITEMIDLIST (item identifier list) structure that is used to identify the icon to be referenced.

UINT uFlags - input
    Specification of the type of icon index to retrieve (defined above). The following flag types are allowed:

    GII_ICON

    GII_SMALLICON

    GII_OPENICON

int * piIndex - output
    Address of the integer that will receive the icon index.

Return Codes

The following list shows common return values:

CWB_OK
    Successful completion.
**CWBUN_INVALID_FLAG_VALUE**

Not a valid supported flag value.

**Usage**

**cwbUN_GetIndexFromItemId**

**Purpose**

Extracts the item index field from a Unity item identifier.

**Syntax**

```c
CWBAPI unsigned int WINAPI cwbUN_GetIndexFromItemId(
    const char * szItemId,
    ULONG* piIndex
);
```

**Parameters**

- `const char * szItemId - input`
  
  The Unity item identifier from which the item index will be extracted.

- `ULONG* piIndex - output`
  
  Address of an unsigned long integer that will receive the item index.

**Return Codes**

The following list shows common return values:

- **CWB_OK**
  
  Successful completion.

- **CWBUN_FORMAT_NOT_VALID**
  
  Specified item identifier not valid.

**Usage**

**cwbUN_GetIndexFromName**

**Purpose**

Extracts the item index field from a fully-qualified Unity object name.

**Syntax**

```c
CWBAPI unsigned int WINAPI cwbUN_GetIndexFromName(
    const char * szObjectName,
    ULONG* piIndex
);
```

**Parameters**

- `const char * szObjectName - input`
  
  The Unity object name from which the item index will be extracted.

- `ULONG* piIndex - output`
  
  Address of an unsigned long integer that will receive the item index.
Return Codes

The following list shows common return values:

**CWB_OK**
Successful completion.

**CWBUN_FORMAT_NOT_VALID**
Specified object name is not valid.

Usage

cwbUN_GetIndexFromPidl

Purpose

Extracts the item index field from a fully-qualified Unity item identifier list.

Syntax

```c
CWBAPI unsigned int WINAPI cwbUN_GetIndexFromPidl(  
    LPCITEMIDLIST pidl,  
    ULONG* piIndex
);
```

Parameters

**LPCITEMIDLIST pidl - input**
Pointer to an ITEMIDLIST (item identifier list) structure from which the item index will be extracted.

**ULONG* piIndex - output**
Address of an unsigned long integer that will receive the item index.

Return Codes

The following list shows common return values:

**CWB_OK**
Successful completion.

**CWBUN_FORMAT_NOT_VALID**
The specified item identifier list is not valid.

Usage

cwbUN_GetLdapPublishCount

Purpose

Returns the number of publishing records configured for this iSeries server. A publish record identifies a category of information to be published, and how and where it is to be published.

Syntax

```c
int cwbUN_GetLdapPublishCount(  
    cwbUN_ldapPubHandle handle,  
    int *count
);
```
Parameters

`cwbUN_LdapPubHandle handle - input`
A handle previously obtained by a call to `cwbUN_OpenLdapPublishing()`.

`int * count - output`
The number of publish records configured on the iSeries server.

Return Codes

The following list shows common return values:

**CWB_OK**
Successful completion.

**CWB_INVALID_API_HANDLE**
Invalid handle.

**CWB_INVALID_POINTER**
A NULL pointer was specified.

Usage

### `cwbUN_GetLdapPublishParentDn`

**Purpose**

Returns the parent distinguished name of the published objects. For example, if the parentDN for publishing users was `cn=users,o=ace industry,c=us`, and user information was published for John Smith, the dn of the published object could be `cn=John Smith,cn=users,ou=ace industry,c=us`.

**Syntax**

```c
int cwbUN_GetLdapPublishParentDnW
( cwbUN_LdapPubHandle handle,
  int index,
  LPWSTR parentDn,
  int *length
);

int cwbUN_GetLdapPublishParentDnA
( cwbUN_LdapPubHandle handle,
  int index,
  LPSTR parentDn,
  int *length
);

int cwbUN_GetLdapPublishParentDn8 /* return parentDn in UTF-8 */
( cwbUN_LdapPubHandle handle,
  int index,
  LPSTR parentDn,
  int *length
);
```

**Parameters**

**cwbUN_LdapPubHandle handle - input**
A handle previously obtained by a call to `cwbUN_OpenLdapPublishing()`.

**int index - input**
Zero-based index of the publish record. This value must be less than the count returned by `cwbUN_GetLdapPublishCount()`.
LPSTR parentDn - output
    Pointer to the buffer that will contain the name of the parentDn.

int * length - input/output
    Pointer to the length of the parentDn buffer. If the buffer is too small to hold the string, including
    space for the terminating NULL, the size of the buffer needed will be filled into this parameter.

Return Codes

The following list shows common return values:

CWB_OK
    Successful completion.

CWB_INVALID_API_HANDLE
    Invalid handle.

CWB_INVALID_API_PARAMETER
    Invalid index.

CWB_INVALID_POINTER
    A NULL pointer was specified.

CWB_BUFFER_OVERFLOW
    The suffix buffer is not large enough to hold the entire result.

Usage

cwbUN_GetLdapPublishPort

Purpose

Returns the port number of the server used to publish this information.

Syntax

int cwbUN_GetLdapPublishPort
    ( cwbUN_ldapPubHandle handle,
      int index,
      int *port,
      cwbUN_LdapCnnSecurity *connectionSecurity
    );

Parameters

cwbUN_ldapPubHandle handle - input
    A handle previously obtained by a call to cwbUN_OpenLdapPublishing().

int index - input
    Zero-based index of the publish record. This value must be less than the count returned by
    cwbUN_GetLdapPublishCount().

int * port - output
    The port number used to connect to the server.

cwbUN_LdapCnnSecurity * connectionSecurity - output
    The type of connection used to connect to the server. This indicates the type of connection that
    can be established over the associated port. Possible values include:

    CWBUN_LDAPCNN_NORMAL
    A normal connection is used.
CWBUN_LDAPCNN_SSL
An SSL connection is used.

Return Codes

The following list shows common return values:

CWB_OK
Successful completion.

CWB_INVALID_API_HANDLE
Invalid handle.

CWB_INVALID_API_PARAMETER
Invalid index.

CWB_INVALID_POINTER
A NULL pointer was specified.

Usage
cwbUN_GetLdapPublishServer

Purpose

Returns the name of the server to which this information is published.

Syntax

```c
int cwbUN_GetLdapPublishServerW
    ( cwbUN_ldapPubHandle handle,
    int index,
    LPWSTR server,
    int *length
    );

int cwbUN_GetLdapPublishServerA
    ( cwbUN_ldapPubHandle handle,
    int index,
    LPSTR server,
    int *length
    );
```

Parameters
cwbUN_ldapPubHandle handle - input
A handle previously obtained by a call to cwbUN_OpenLdapPublishing().

int index - input
Zero-based index of the publish record. This value must be less than the count returned by cwbUN_GetLdapPublishCount().

LPSTR server - output
Pointer to the buffer that will contain the name of the server.

int * length - input/output
Pointer to the length of the server buffer. If the buffer is too small to hold the string, including space for the terminating NULL, the size of the buffer needed will be filled into this parameter.

Return Codes
The following list shows common return values:

**CWB_OK**
Successful completion.

**CWB_INVALID_API_HANDLE**
Invalid handle.

**CWB_INVALID_API_PARAMETER**
Invalid index.

**CWB_INVALID_POINTER**
A NULL pointer was specified.

**CWB_BUFFER_OVERFLOW**
The suffix buffer is not large enough to hold the entire result.

**Usage**

**cwbUN_GetLdapPublishType**

**Purpose**
Returns the type of information for which this publish record is.

**Syntax**

```c
int cwbUN_GetLdapPublishType
    ( cwbUN_ldapPubHandle handle,
    int index,
    cwbUN_LdapPubCategories *information
    );
```

**Parameters**

**cwbUN_ldapPubHandle handle** - input
A handle previously obtained by a call to cwbUN_OpenLdapPublishing().

**int index** - input
Zero-based index of the publish record. This value must be less than the count returned by cwbUN_GetLdapPublishCount().

**cwbUN_LdapPubCategories * information** - output
The type of information for which this publish record is. Possible values include:

- **CWBUN_LDAP_PUBLISH_USERS**
  User information
- **CWBUN_LDAP_PUBLISH_COMPUTERS**
  iSeries systems
- **CWBUN_LDAP_PUBLISH_NETWORK_INVENTORY**
  NetFinity
- **CWBUN_LDAP_PUBLISH_PRINTERS**
  iSeries printers

**Return Codes**

The following list shows common return values:
**CWB_OK**  
Successful completion.

**CWB_INVALID_API_HANDLE**  
Invalid handle.

**CWB_INVALID_API_PARAMETER**  
Invalid index.

**CWB_INVALID_POINTER**  
A NULL pointer was specified.

**Usage**

**cwbUN_GetLdapServerBindDn**

**Purpose**

Returns the distinguished name used by Operations Navigator to bind to the server. If no name is configured, an empty string will be returned.

**Syntax**

```c
int cwbUN_GetLdapServerBindDnA
  ( cwbUN_ldapBindInfoHandle handle,
    LPSTR bindDn,
    int *length
  );

int cwbUN_GetLdapServerBindDnW
  ( cwbUN_ldapBindInfoHandle handle,
    LPWSTR bindDn,
    int *length
  );

int cwbUN_GetLdapServerBindDn8 /* returns bindDn in UTF-8 */
  ( cwbUN_ldapBindInfoHandle handle,
    LPSTR bindDn,
    int *length
  );
```

**Parameters**

**cwbUN_ldapBindInfoHandle handle - input**  
A handle previously obtained by a call to cwbUN_OpenLdapBindInfo().

**LPSTR bindDn - output**  
Pointer to the buffer that will contain the distinguished name used by Operations Navigator to bind to the server.

**int * length - input/output**  
Pointer to the length of the bindDn buffer. If the buffer is too small to hold the string, including space for the terminating NULL, the size of the buffer needed will be filled into this parameter.

**Return Codes**

The following list shows common return values:

**CWB_OK**  
Successful completion.

**CWB_INVALID_POINTER**  
NULL pointer was specified.
CWB_BUFFER_OVERFLOW
The suffix buffer is not large enough to hold the entire result.

Usage

**cwbUN_GetLdapSvrPort**

**Purpose**

Returns the port number that is used by the LDAP server.

**Syntax**

```c
int cwbUN_GetLdapSvrPort(
    cwbUN_ldapSvrHandle handle,
    int *port,
    int *sslPort
);
```

**Parameters**

- **cwbUN_ldapSvrHandle handle - input**
  A handle previously obtained by a call to `cwbUN_OpenLocalLdapServer()`.
- **int * port - output**
  The port number used for LDAP connections.
- **int * sslPort - output**
  The port number used for SSL connections.

**Return Codes**

The following list shows common return values:

- **CWB_OK**
  Successful completion.
- **CWB_INVALID_API_HANDLE**
  Invalid handle.
- **CWB_INVALID_POINTER**
  A NULL pointer was specified.

Usage

**cwbUN_GetLdapSvrSuffixCount**

**Purpose**

Returns the number of suffixes configured for this server. A suffix is the distinguished name (DN) of a starting point in the directory tree.

**Syntax**

```c
int cwbUN_GetLdapSvrSuffixCount(
    cwbUN_ldapSvrHandle handle,
    int *count
);
```
Parameters

cwbUN_ldapSvrHandle handle - input
   A handle previously obtained by a call to cwbUN_OpenLocalLdapServer().

int * count - output
   The number of suffixes present on the server.

Return Codes

The following list shows common return values:

CWB_OK
   Successful completion.

CWB_INVALID_API_HANDLE
   Invalid handle.

CWB_INVALID_POINTER
   A NULL pointer was specified.

Usage

cwbUN_GetLdapSvrSuffixName

Purpose

Returns the distinguished name of the suffix.

Syntax

int cwbUN_GetLdapSuffixNameA
   ( cwbUN_ldapSvrHandle handle,
     int index,
     LPSTR suffix,
     int *length
   );

int cwbUN_GetLdapSuffixNameW
   ( cwbUN_ldapSvrHandle handle,
     int index,
     LPWSTR suffix,
     int *length
   );

int cwbUN_GetLdapSuffixName8 /* returns suffix in UTF-8 */
   ( cwbUN_ldapSvrHandle handle,
     int index,
     LPSTR suffix,
     int *length
   );

Parameters

cwbUN_ldapSuffixHandle handle - input
   A handle previously obtained by a call to cwbUN_OpenLocalLdapServer().

int index - input
   Zero-based index of the suffix. This value must be less than the count returned by
   cwbUN_GetLdapSvrSuffixCount().

LPSTR suffix - output
   Pointer to the buffer that will contain the distinguished name of the suffix.
int * length - input/output

Pointer to the length of the suffix buffer. If the buffer is too small to hold the string, including space for the terminating NULL, the size of the buffer needed will be filled into this parameter.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Invalid handle.

CWB_INVALID_API_PARAMETER

Invalid index.

CWB_INVALID_POINTER

A NULL pointer was specified.

CWB_BUFFER_OVERFLOW

The suffix buffer is not large enough to hold the entire result.

Usage

cwbUN_GetListObject

Purpose

Get a pointer to the object associated with the specified list object name.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetListObject(
    const char * szFileName,
    LPVOID *pListObject
);

Parameters

const char * szFileName - input

The Unity object name from which the object pointer will be found and returned.

LPVOID pListObject - output

Address of a pointer to the request Unity object.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

Usage

cwbUN_GetODBCConnection

Purpose

Return the handle to an ODBC connection on the specified iSeries server server system. If no connection exists to the specified iSeries server, the API obtains a new handle.
Syntax

```
CWBAPI unsigned int WINAPI cwbUN_GetODBCConnection(
    const char * szSystemName,
    HDBC *phDBC
);
```

Parameters

**const char * szSystemName - input**
The name of the iSeries system on which to retrieve an ODBC connection.

**HDBC *phDBC - output**
Address to return the ODBC connection handle.

Return Codes

The following list shows common return values:

**CWB_OK**
Successful completion.

Usage

cwbUN_GetParentFolderNameFromName

Purpose

Extracts the name of an object’s parent folder from a fully-qualified Unity object name.

Syntax

```
CWBAPI unsigned int WINAPI cwbUN_GetParentFolderNameFromName(
    const char * szObjectName,
    char * szParentFolderName,
    UINT cchMax
);
```

Parameters

**const char * szObjectName - input**
The Unity object name from which the parent folder name will be extracted.

**char * szParentFolderName - output**
Address of the buffer that will receive the null-terminated parent folder name.

**UINT cchMax - input**
Size of the buffer that will receive the null-terminated parent folder name.

Return Codes

The following list shows common return values:

**CWB_OK**
Successful completion.

**CWBUN_FORMAT_NOT_VALID**
Specified object name is not valid.
The buffer is too small to contain the returned string.

Usage

cwbUN_GetParentFolderPathFromName

Purpose

Given a fully-qualified Unity object name, returns the fully-qualified object name of the object’s parent folder.

Syntax

```c
CWBAPI unsigned int WINAPI cwbUN_GetParentFolderPathFromName(
   const char * szObjectName,
   char * szParentFolderPath,
   UINT cchMax
);
```

Parameters

- **const char * szObjectName** - input
  The Unity object name from which the parent folder object name will be extracted.

- **char * szParentFolderPath** - output
  Address of the buffer that will receive the null-terminated parent folder object name.

- **UINT cchMax** - input
  Size of the buffer that will receive the null-terminated parent folder object name.

Return Codes

The following list shows common return values:

**CWB_OK**
Successful completion.

**CWBUN_FORMAT_NOT_VALID**
Specified object name is not valid.

**CWB_BUFFER_OVERFLOW**
The buffer is too small to contain the returned string.

Usage

cwbUN_GetParentFolderPidl

Purpose

Given a fully-qualified Unity item identifier list, returns the fully-qualified item identifier list of the object’s parent folder.

Syntax

```c
CWBAPI unsigned int WINAPI cwbUN_GetParentFolderPidl(
   LPCITEMIDLIST pidl,
   LPITEMIDLIST *ppidl
);
```
Parameters

LPCITEMIDLIST pidl - input
    Pointer to an ITEMIDLIST (item identifier list) structure from which the parent folder item identifier list will be extracted.

LPITEMIDLIST* ppidl - output
    Address of an item identifier list pointer that will receive the parent folder item identifier list.

Return Codes

The following list shows common return values:

CWB_OK
    Successful completion.

CWBUN_FORMAT_NOT_VALID
    The specified item identifier list is not valid.

Usage

cwbUN_GetResourceLibraryPath

Purpose

Determines which of the installed languages is supported by the specified Operations Navigator plug-in, and returns the directory pathname of the subdirectory that contains the plug-in’s NLS resources for the currently active language. If the plug-in does not support the active language, the pathname for a related language that is supported will be returned, if one can be found. Plug-in developers should use this API to determine from which directory they need to load their NLS resources.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetResourceLibraryPath(
    const char * pszProgID,
    char * pszLibraryPath,
    UINT cchMax
);

Parameters

const char * pszProgID - input
    The programmatic identifier which uniquely identifies an Operations Navigator plug-in. It should be of the form <vendor><component>.

char * pszLibraryPath - output
    Address of the buffer that will receive the fully-qualified pathname of the NLS directory.

UINT cchMax - input
    Size of the buffer that will receive the fully-qualified pathname of the NLS directory.

Return Codes

The following list shows common return values:

CWB_OK
    Successful completion.

CWBUN_FORMAT_NOT_VALID
    Specified ProgID is not valid.
CWBUN_RTN_STR_TOO_LONG
The buffer is too small to contain the returned string.

CWBUN_LANG_NOT_FOUND
The specified plug-in does not support any of the languages that are currently installed on the client.

Usage

cwbUN_GetSharedImageList
Purpose
Retrieve the icon image list associated with Operations Navigator.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetSharedImageList(
    UINT uFlags,
    HIMAGELIST *phImageList
);

Parameters

UINT uFlags - input
Specification of the type of image list to retrieve (defined above) The following flag types are allowed:

    GSIL_ICON

    GSIL_SMALLICON

HIMAGELIST* phImageList - Address of the variable that will receive the image list handle.

Return Codes

The following list shows common return values:

CWB_OK
Successful completion.

CWBUN_INVALID_FLAG_VALUE
Not a valid supported flag value.

CWBUN_CANT_GET_IMAGELIST
A failure occurred while attempting to get the icon image list.

Usage

cwbUN_GetSystemHandle
Purpose
Returns a system handle that contains the security (SSL), userID, and password settings that are used for this iSeries system. The system handle will have the settings that were configured in Operations Navigator for the input iSeries system name.
If the application name is set to NULL, the returned system handle will be unique. If the application name is set, the same system handle that matches the application name will be returned.

If an application requires a unique OS/400 job for a system, then NULL or a unique name should be passed in for the application name.

If an application requires to share an OS/400 job, then all callers of this function should pass in the same application name.

Syntax

```c
CWBAPI unsigned int WINAPI cwbUN_GetSystemHandle(
    char * szSystemName,
    char * szAppName,
    cwbCO_SysHandle * systemHandle
);
```

Parameters

- `char * szSystemName - input`
  Pointer to an ASCIIZ string that contains the name of the system for which you want a system handle to be created.

- `char * szAppName - input`
  Pointer to an ASCIIZ string of no more than 12 characters. This uniquely identifies the application that will share a single system handle.

- `cwbCO_SysHandle * systemHandle - output`
  Pointer to the handle of the system for this system name.

Return Codes

The following list shows common return values:

- **CWB_OK**
  Successful completion.

- **CWBUN_NULL_PARM**
  System name was NULL.

- **CWBUN_INVALID_NAME_PARM**
  The system name is not valid.

- **CWB_NON_REPRESENTABLE_UNICODE_CHAR**
  One or more input UNICODE characters have no representation in the code page that is being used.

- **CWB_API_ERROR**
  The system handle could not be returned.

Usage

This function must be used by all third-party applications that want to support SSL using the Client Access APIs. For example, all Client Access communications APIs require a system handle to support SSL.

When the caller of this function no longer needs the system handle for communications, the handle can be released by calling function `cwbUN_ReleaseSystemHandle`.

All handles will be released when the Operations Navigator application (cwbunnnav.exe) terminates.
cwbUN_GetSystemNameFromName

Purpose

Extracts the iSeries system name from a fully-qualified Unity object name.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetSystemNameFromName(
    const char * szObjectName,
    char * szSystemName,
    UINT cchMax
);

Parameters

const char * szObjectName - input
    The Unity object name from which the system name will be extracted.

char * szSystemName - output
    Address of the buffer that will receive the null-terminated system name.

UINT cchMax - input
    Size of the buffer that will receive the null-terminated system name.

Return Codes

The following list shows common return values:

CWB_OK
    Successful completion.

CWBUN_FORMAT_NOT_VALID
    Specified object name is not valid.

CWB_BUFFER_OVERFLOW
    The buffer is too small to contain the returned string.

Usage

cwbUN_GetSystemNameFromPidl

Purpose

Extracts the iSeries system name from a fully-qualified Unity item identifier list.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetSystemNameFromPidl(
    LPCITEMIDLIST pidl,
    char * szSystemName,
    UINT cchMax
);

Parameters

LPCITEMIDLIST pidl - input
    Pointer to an ITEMIDLIST (item identifier list) structure from which the system name will be extracted.
char * szSystemName - output
Address of the buffer that will receive the null-terminated system name.

UINT cchMax - input
Size of the buffer that will receive the null-terminated system name.

Return Codes
The following list shows common return values:

CWB_OK
Successful completion.

CWBUN_FORMAT_NOT_VALID
The specified item identifier list is not valid.

CWB_BUFFER_OVERFLOW
The buffer is too small to contain the returned string.

Usage
cwbUN_GetSystemValue
Purpose
Returns a string that contains the value of an iSeries system value.

Syntax
CWBAPI unsigned int WINAPI cwbUN_GetSystemValue(
    USHORT usSystemValueId,
    const char * szSystemName,
    char * szSystemValue,
    UINT cchMax
);

Parameters
const char * szSystemValueId - input
A numeric value that identifies the iSeries system value to be retrieved. Definitions for the system value constants are in the header file 'CWBA4SVL.H'

char * szSystemValue - output
Address of the buffer that will receive the null-terminated system value string.

UINT cchMax - input
Size of the buffer that will receive the null-terminated value string.

Return Codes
The following list shows common return values:

CWB_OK
Successful completion.

CWBUN_INTERNAL_ERROR
Could not retrieve the system value.

CWB_BUFFER_OVERFLOW
The buffer is too small to contain the returned string.
Usage

The value that is returned by this API is not a National Language Support (NLS) string and is not translated. For example, "NONE" will be returned instead of 'None.'

cwbUN_GetTypeFromItemId

Purpose

Extracts the item type field from a Unity item identifier.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetTypeFromItemId(
    const char * szItemId,
    char * szType,
    UINT cchMax
);

Parameters

const char * szItemId - input
    The Unity item identifier from which the item type will be extracted.

cchar * szType - output
    Address of the buffer that will receive the null-terminated item type.

UINT cchMax - input
    Size of the buffer that will receive the null-terminated item type.

Return Codes

The following list shows common return values:

CWB_OK
    Successful completion.

CWBUN_FORMAT_NOT_VALID
    Specified item identifier not valid.

CWB_BUFFER_OVERFLOW
    The buffer is too small to contain the returned string.

Usage

cwbUN_GetTypeFromName

Purpose

Extracts the item type field from a fully-qualified Unity object name.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetTypeFromName(
    const char * szObjectName,
    char * szType,
    UINT cchMax
);

Parameters

const char * szObjectName - input

The Unity object name from which the item index will be extracted.

char * szType - output

Address of the buffer that will receive the null-terminated item type.

UINT cchMax - input

Size of the buffer that will receive the null-terminated item type.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWBUN_FORMAT_NOT_VALID

Specified object name is not valid.

CWB_BUFFER_OVERFLOW

The buffer is too small to contain the returned string.

Usage

cwbUN_GetTypeFromPidl

Purpose

Extracts the item index field from a fully-qualified Unity item identifier list.

Syntax

CWBAPI unsigned int WINAPI cwbUN_GetTypeFromPidl(
    LPCITEMIDLIST pidl,
    char * szType,
    UINT cchMax
);

Parameters

LPCITEMIDLIST pidl - input

Pointer to an ITEMDLIST (item identifier list) structure from which the item index will be extracted.

char * szType - output

Address of the buffer that will receive the null-terminated item type.

UINT cchMax - input

Size of the buffer that will receive the null-terminated item type.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWBUN_FORMAT_NOT_VALID

The specified item identifier list is not valid.

CWB_BUFFER_OVERFLOW

The buffer is too small to contain the returned string.
Usage

**cwbUN_GetUserAttribute**

**Purpose**

Returns a string that contains the value of a user profile attribute for the current Operations Navigator user.

**Syntax**

```c
CWBAPI unsigned int WINAPI cwbUN_GetUserAttribute(
    USHORT usAttributeId,
    const char * szSystemName,
    char * szValue,
    UINT cchMax
);
```

**Parameters**

- **USHORT usAttributeId - input**
  A numeric value which identifies the user attribute value to be retrieved. Definitions for the user attribute constants are in the header file 'CWBA4USR.H'.

- **const char * szSystemName - input**
  The name of the iSeries system from which to retrieve the user attribute.

- **char * szValue - output**
  Address of the buffer that will receive the null-terminated attribute value string.

- **UINT cchMax - input**
  Size of the buffer that will receive the null-terminated value string.

**Return Codes**

The following list shows common return values:

- **CWB_OK**
  Successful completion.

- **CWBUN_INTERNAL_ERROR**
  Could not retrieve attribute value.

- **CWB_BUFFER_OVERFLOW**
  The buffer is too small to contain the returned string.

**Usage**

The value that is returned by this API is **not** an NLS string and is not translated. For example, ‘*NONE*’ will be returned.

**Return codes unique to Operations Navigator APIs**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6000</td>
<td>CWBUN_BAD_PARAMETER</td>
</tr>
<tr>
<td></td>
<td>An input parameter was not valid.</td>
</tr>
<tr>
<td>6001</td>
<td>CWBUN_FORMAT_NOT_VALID</td>
</tr>
<tr>
<td></td>
<td>The input object name was not valid.</td>
</tr>
<tr>
<td>6002</td>
<td>CWBUN_WINDOW_NOTAVAIL</td>
</tr>
<tr>
<td></td>
<td>View window not found.</td>
</tr>
<tr>
<td>6003</td>
<td>CWBUN_INTERNAL_ERROR</td>
</tr>
</tbody>
</table>

Chapter 5. Plug-in developers reference  79
Processing error occurred.

6004  CWBUN_USER_NOT_AUTHORIZED
User does not have specified authority.

6005  CWBUN_OBJECT_NOT_FOUND
Object not found on the AS/400.

6006  CWBUN_INVALID_ITEM_ID
Invalid item ID parameter.

6007  CWBUN_NULL_PARM
NULL parameter passed.

6008  CWBUN_RTN_STR_TOO_LONG
String too long for return buffer.

6009  CWBUN_INVALID_OBJ_NAME
Invalid object name parameter.

6010  CWBUN_INVALID_PIDL
Invalid PIDL parameter.

6011  CWBUN_NULL_PIDL_RETURNED
Parent folder PIDL was NULL.

6012  CWBUN_REFRESH_FAILED
Refresh list failed.

6012  CWBUN_UPDATE_FAILED
Update toolbar failed.

6013  CWBUN_INVALID_NAME_TYPE
Invalid AS/400 name type.

6014  CWBUN_INVALID_AUTH_TYPE
Invalid authority type.

6016  CWBUN_HOST_COMM_ERROR
AS/400 communications error.

6017  CWBUN_INVALID_NAME_PARM
Invalid name parameter.

6018  CWBUN_NULL_DISPLAY_STRING
Null display string returned.

6019  CWBUN_GENERAL_FAILURE
General AS/400 operation failure.

6020  CWBUN_INVALID_SYSVAL_ID
Invalid system value ID.

6021  CWBUN_INVALID_LIST_OBJECT
Can not get list object from name.

6022  CWBUN_INVALID_IFS_PATH
Invalid IFS path specified.

6023  CWBUN_LANG_NOT_FOUND
Extension does not support any of the languages installed.

6024  CWBUN_INVALID_USER_ATTR_ID
Invalid user attribute ID.

6025  CWBUN_GET_USER_ATTR_FAILED
Unable to retrieve user attribute.

6026  CWBUN_INVALID_FLAG_VALUE
Invalid flag parameter value set.

6027  CWBUN_CANT_GET_IMAGELIST
Cannot get icon image list.

The following return codes are for name check APIs.

6050  CWBUN_NAME_TOO_LONG
Name is too long.

6051  CWBUN_NAME_NULLSTRING
String in empty - no chars at all.
The following return codes are for LDAP-related APIs.

6101 CWBUN_LDAP_NOT_AVAIL
LDAP is not installed or configured.
6102 CWBUN_LDAP_BIND_FAILED
LDAP bind failed.

The following return codes are for check AS/400 name APIs.

1001 CWBUN_NULLSTRING
String is empty.
1004 CWBUN_INVALIDCHAR
Invalid character.
1005 CWBUN_STRINGTOOLONG
String is too long.
1006 CWBUN_MISSINGENDQUOTE
End quote for quoted string missing.
1007 CWBUN_INVALIDQUOTECHAR
Character invalid for quoted string.
1008 CWBUN_ONLYBLANKS
String contains only blanks.
1009 CWBUN_STRINGTOOSHORT
String is less than the defined minimum.
1011 CWBUN_TOOLONGFORIBM
String is OK, but too long for IBM commands.
1012 CWBUN_INVALIDFIRSTCHAR
First character is invalid.
1999 CWBUN_GENERALFAILURE
Unspecified error.
Operations Navigator structure and flow of control for Visual Basic plug-ins

For Visual Basic plug-ins, Operations Navigator provides a built-in ActiveX server that manages the communication between Navigator and the plug-in’s implementation. Visual Basic programmers who are developing Operations Navigator plug-ins then use the facilities that are provided by Microsoft’s Visual Basic 5.0 to create their plug-in classes, and to package them in an ActiveX server DLL.

Plug-ins work by responding to method calls from Operations Navigator that are generated in response to user actions. For example, when a user right-clicks on an object in the Navigator hierarchy, Navigator constructs a context menu for the object and displays the menu on the screen. Navigator obtains the menu items by calling each plug-in that has registered its intent to supply context menu items for the selected object type.

The functions that are implemented by a plug-in are logically grouped into interfaces. An interface is a set of logically related methods on a class that Operations Navigator can call to perform a specific function. For Visual Basic plug-ins, three interfaces are defined:

- ListManager
- ActionsManager
- DropTargetManager

Operations Navigator data for Visual Basic plug-ins

When the Navigator calls a function implemented by a plug-in, the request typically involves an object or objects the user selected in the main Navigator window. The plug-in must be able to determine which objects have been selected. The plug-in receives this information as a list of fully-qualified object names. For Visual Basic plug-ins, an ObjectName class is defined that provides information about the selected objects. Plug-ins that add folders to the object hierarchy must return items in the folder to Operations Navigator in the form of “item identifiers.” For Visual Basic plug-ins, an ItemIdentifier class is defined that is used by the plug-in to return the requested information.

Operations Navigator services for Visual Basic plug-ins

An Operations Navigator plug-in sometimes will need to affect the behavior of the main Navigator window. For example, following completion of a user operation, it may be necessary to refresh the Navigator list view or to insert text into the Navigator’s status area. A utility class called UIServices is supplied in the Visual Basic environment that provides the required services. A Visual Basic plug-in also can use the C++ APIs in the cw bun.h header file to achieve similar results. For detailed descriptions of this class and its methods, see the online help that is provided with the Operations Navigator Visual Basic Plug-in Support DLL (cw bunv hi.dll and cw bunv hi.hlp).

Operations Navigator Visual Basic interfaces

A Visual Basic plug-in must implement one or more Operations Navigator interface classes, depending on the type of function that the developer intends to provide to the Operations Navigator.

The Client Access Express Toolkit contains a link to the Visual Basic interface definition help file.

There are three Operations Navigator interface classes:

- Operations Navigator ListManager interface class
- Operations Navigator ActionsManager interface class
- Operations Navigator DropTargetManager interface class

Your application does not have to implement all three interface
Operations Navigator ListManager interface class
The ListManager interface class is used for data serving in Operations Navigator. For example, when a list view needs to be created and filled with objects, Operations Navigator will call methods in the ListManager class to do this. The Visual Basic Sample plug-in provides an example of this class in the file listman.cls. You must have a ListManager class if your plug-in needs to populate Operations Navigator component lists.

For detailed descriptions of this class and its methods, see the online help provided with the Operations Navigator Visual Basic Plug-in Support DLL (cwbnvbi.dll and cwbnvbi.hlp).

Operations Navigator ActionsManager interface class
The ActionsManager interface class is used to build context menus, and to implement commands of the context menu actions. For example, when a user performs a right mouse-click on a Visual Basic list object in Operation Navigator, the queryActions method in the ActionsManager interface class will be called to return the context menu item strings. The Visual Basic Sample plug-in provides an example of this class in the file actnman.cls. You must define an ActionsManager interface class for each unique object type that your plug-in supports. You can specify the same ActionsManager interface class for different object types, but your code logic must handle being called with multiple types of objects.

For detailed descriptions of this class and its methods, see the online help provided with the Operations Navigator Visual Basic Plug-in Support DLL (cwbnvbi.dll and cwbnvbi.hlp).

Operations Navigator DropTargetManager interface class
The DropTargetManager interface class is used to handle drag-and-drop operations in Operations Navigator. When a user selects a Visual Basic list object, and performs mouse drag-and-drop operations on it, methods in this class will be called to perform the drag-and-drop operations.

For detailed descriptions of this class and its methods, see the online help provided with the Operations Navigator Visual Basic Plug-in Support DLL (cwbnvbi.dll and cwbnvbi.hlp).

Operations Navigator structure and flow of control for Java plug-ins
For Java plug-ins, Operations Navigator provides a built-in ActiveX server that manages the communication between the Navigator and the plug-in’s Java classes. The server component uses the Java Native Interface (JNI) API to create the plug-in’s objects and to call their methods. Thus, Java programmers who are developing Operations Navigator plug-ins do not need to be concerned with the details of ActiveX server implementation.

When a user is interacting with Operations Navigator Java plug-ins, calls will be generated to the different registered Java interface classes for the implementation of the specific request.

Plug-ins work by responding to method calls from Operations Navigator that are generated in response to user actions. For example, when a user right-clicks on an object in the Navigator hierarchy, the Navigator constructs a context menu for the object, and displays the menu on the screen. The Navigator obtains the menu items by calling each plug-in that has registered its intent to supply context menu items for the selected object type.

The functions that are implemented by a plug-in logically are grouped into “interfaces.” An interface is a set of logically related methods on a class that Operations Navigator can call to perform a specific function. For Java plug-ins, the following three Java interfaces are defined:

- ListManager
- ActionsManager
- DropTargetManager
**Product architecture for Operations Navigator plug-ins**

The internal architecture of the Operations Navigator product reflects that it is intended to serve as an integration point for an extensible, broad-based operations interface for the iSeries server. Each functional component of the interface is packaged as an ActiveX server. The Navigator learns about the existence of a particular server component by means of entries in the Windows registry. Multiple servers may register their request to add menu items and dialogs to a given object type in the Navigator hierarchy.

**Note:** For third-party Java plug-ins to be available to Operations Navigator users, Client Access users must have Version 4 Release 4 Modification Level 0 of Client Access Express for Windows installed on their personal computers.

**Operations Navigator data for Java plug-ins**

When the Navigator calls a function implemented by a plug-in, the request typically involves an object or objects the user selected in the main Navigator window. The plug-in must be able to determine which objects have been selected. The plug-in receives this information as a list of fully-qualified object names. For Java plug-ins, an ObjectName class is defined that provides information about the selected objects. Plug-ins that add folders to the object hierarchy must return items in the form of “item identifiers.” For Java plug-ins, an ItemIdentifier class is defined that is used by the plug-in to return the requested information.

An Operations Navigator plug-in sometimes will need to affect the behavior of the main Navigator window. For example, following completion of a user operation, it may be necessary to refresh the Navigator list view or to insert text into the Navigator’s status area. Utility classes are supplied in the package com.ibm.as400.opnav that provide the required services.
Chapter 6. Customize the plug-in registry files

Registry files identify plug-ins to Operations Navigator, describe their functions, and specify any prerequisites for using the plug-in. The sample plug-ins include two registry files: a windows-readable copy for use during development, and a copy for distribution on the iSeries server. You'll need to make some modifications to these registry files after developing your plug-in. To help you make those changes, this topic provides an overview of the registry files, and detailed descriptions of the required sections of each registry file.

Operations navigator uses the registry files to learn about the plug-ins existence, requirements and functions. In order to provide that information every plug-in must specify at least the following information:

- A "primary" registry key that that provides global information about the plug-in. This section includes the Programmatic Identifier (ProgID) which specifies the vendor and component name for your plug-in, and will also name the folder in which your plug-in resides on the iSeries server. The ProgID must follow the form <vendor>.<component>, i.e. IBM.Sample.
- Registry keys that identify the object types in the Operations Navigator hierarchy for which a plug-in intends to supply additional function.
- A separate registry key for the root of each sub tree of objects that a plug-in adds to the object hierarchy. This key contains information about the root folder of the sub tree.

Descriptions of the required sections of the registry files, and the recommended changes:

- C++ registry files
- VB registry files
- Java registry files

Special considerations for the registry files

- Property sheet handling in C++
- Property sheet handling in VB
- SSL support in plug-ins

Customize the C++ registry values

The sample plug-includes two registry files: SAMDBG.REG, a windows-readable registry file for use during development and SAMPRLS.REG, a registry file for distribution on the iSeries server. The following table describes the sections in these registry files, and recommends changes for use when developing your own plug-in.

Primary registry key

```c++
; Define the primary registry key for the plugin
; NOTE: NLS and ServerEntryPoint DLL names must
; not contain qualified directory paths

[HKLM\SOFTWARE\IBM.AS400.Network\3RD PARTY plug-inS\IBM.Sample]
"Type"="PLUGIN"
"NLS"="sampmri.dll"
"NameID"=dword:00000080
"DescriptionID"=dword:00000081
"MinimumIMPIRelease"="NONE"
"MinimumRISCRelease"="030701"
"ProductID"="NONE"
"ServerEntryPoint"="sampext.dll"
```
Data Server Implementation

[This section will register an IA4HierarchyFolder implementation for each new folder added to the Operations Navigator hierarchy.]

[REGISTRY]
[HKEY_CLASSES_ROOT\CLSID\{D09970E1-9073-11d0-82BD-08005AA74F5C}\]
@="AS/400 Data Server - Sample Data"

[REGISTRY]
[HKEY_CLASSES_ROOT\CLSID\{D09970E1-9073-11d0-82BD-08005AA74F5C}\InprocServer32\]
@="%CLIENTACCESS%\Plugins\IBM.Sample\sampext.dll"
"ThreadingModel"="Apartment"

1. Change the name of the DLL to match the name of the DLL that is generated by your new project workspace.
2. Generate and copy a new GUID (See the global changes section at the bottom of this page)
3. Replace both occurrences of the CLSID in this section of the registry with the new GUID string you just generated.
4. Search for the string “IMPLEMENT_OLECREATE” in your version of the file SAMPDATA.CPP
5. Paste the new GUID over the existing CLSID in the comment line, then change the CLSID in the IMPLEMENT_OLECREATE macro call to match the hex values in your new GUID. Replace the word "Sample" with the name of your new folder.
6. Create two new source files for each new GUID, using a renamed copy of SAMPDATA.H and SAMPDATA.CPP as a base.
7. Note: The header file (.H) contains the class declaration for the new implementation class. The implementation file (.CPP) contains the code that obtains the data for the new folder.
8. Replace all occurrences of the class name "CSampleData" in the two source files with a class name that is meaningful in the context of your plug-in.
9. To add the new implementation files to the project workspace, open the Insert menu and select Files Into Project....
10. Because you are duplicating SAMPDATA.CPP in this way, all your new folders will initially contain library objects.

Shell plug-in implementation

[REGISTRY]
[HKEY_CLASSES_ROOT\CLSID\{3D7907A1-9080-11d0-82BD-08005AA74F5C}\]
@="AS/400 Shell plug-ins - Sample"

[REGISTRY]
[HKEY_CLASSES_ROOT\CLSID\{3D7907A1-9080-11d0-82BD-08005AA74F5C}\InprocServer32\]
@="%CLIENTACCESS%\Plugins\IBM.Sample\sampext.dll"
"ThreadingModel"="Apartment"

; Approve shell plug-in (required under Windows NT)

[REGISTRY]
[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Shell plug-ins\Approved\]
"{3D7907A1-9080-11d0-82BD-08005AA74F5C}"="AS/400 Shell plug-ins - Sample"

This section registers the shell plug-in implementation class. Every c++ plug-in must use this section.
1. Change the DLL name to match the name of the DLL that was generated by your new project workspace.
2. Generate and copy a new GUID (see the global changes section at the bottom of this page).
3. Replace all occurrences of the CLSID in the entries that are shown in the example above with the new GUID you just generated.
4. Search for the string “IMPLEMENT_OLECREATE” in your version of the file EXTINTFC.CPP.
5. Paste the new GUID over the existing CLSID in the comment line, then change the CLSID in the IMPLEMENT_OLECREATE macro call to match the hex values in your new GUID.

Shell plug-in implementation for objects

```plaintext
; Register a context menu handler for the new folder and its objects
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-ins\IBM.Sample\shellex\Sample]*
\ContextMenuHandlers\{3D7907A1-9080-11d0-82BD-08005AA74F5C}

; Register a property sheet handler for the new folder and its objects
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.Sample\shellex\Sample]*
\PropertySheetHandlers\{3D7907A1-9080-11d0-82BD-08005AA74F5C}

; Register that this plug-in supports Secure Socket Layer (SSL) Connection
; Note: "Support Level"=dword:00000001 says the plugin supports SSL
; Note: "Support Level"=dword:00000000 says the plugin does not support SSL
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.Sample\SSL]
"Support Level"=dword:00000001
```

The final section of the registry specifies which objects in the Navigator hierarchy are affected by implementation of the plug-in.

1. Replace the CLSID in this section with the new GUIDs.
2. If your plug-in will not add additional property pages to a property sheet for a folder or object, then remove the registry entry for the property sheet handler.
3. If your plug-in will not be a drop handler for objects, remove the drag and drop context menu handler and drop handler registry entries.

4. Edit the subkeys \Sample\. For more information see, Shell plug-ins.

5. Edit or remove the code in your version of EXTINTFC.CPP, that checks for the object types defined by the sample. You should see the folders, context menu items, property pages, and drop actions from the sample, depending on how much function from the sample you decided to retain.

Note: The code file based on the sample file EXTINTFC.CPP contains the code that will be called for context menus, property pages, and drop actions. The sample code contains checks for the object types that the sample defines. You must edit this file and either remove these tests or change them to check for the object types for which you wish to provide new function.

Global changes

You have to specify a unique ProgID and GUIDs for use throughout the plug-in registry file.

Define a unique programmatic identifier, or ProgID, for your plug-in:

The ProgID should match the <vendor>.<component> text string, where vendor identifies the name of the vendor who developed the plug-in, and component describes the function being provided. In the sample plug-in, the string "IBM.Sample" identifies IBM as the vendor, and "Sample" as the description of the function that is provided by the plug-in. This will be used throughout the registry file, and will name the directory where your plug-in will reside on both the iSeries server and the workstation. Replace every occurrence of "IBM.Sample" in the registry file with your ProgID.

Generate new GUIDs, and replace the CLSID values in the registry file:

For your Operations Navigator C++ plug-in to work properly, you must replace specific CLSIDs in your new registry file with GUIDs that you generate. The Component Object Model from Microsoft uses 16-byte hex integers to uniquely identify ActiveX implementation classes and interfaces. These integers are known as GUIDs (Globally Unique Identifiers). GUIDs that identify implementation classes are called CLSIDs. (pronounced "class IDs") Operations Navigator uses the Windows ActiveX runtime support to load a plug-in’s components, and to obtain a pointer to an instance of the plug-in’s implementation of a particular interface. A CLSID in the registry uniquely identifies a specific implementation class that resides in a specific ActiveX server DLL. The first stage of this mapping, from the CLSID to the name and location of the server DLL, is accomplished by means of a registry entry. Therefore, an Operations Navigator plug-in must register a CLSID for each implementation class that it provides.

Follow these steps to generate your GUIDs:
1. From the Windows taskbar, select Start and then Run.
2. Type GUIDGEN and click OK.
3. Make sure that Registry Format is selected
4. To generate a new GUID value, select New GUID.
5. To copy the new GUID value to the clipboard, select Copy.

Example: Primary registry key

The primary registry key defines a set of fields that specify global information for the plug-in. This information is required.
Define the primary registry key for the plugin
NOTE: NLS and ServerEntryPoint DLL names must not contain qualified directory paths

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-ins\IBM.Sample]
"Type"="PLUGIN"
"NLS"="sampmri.dll"
"NameID"=dword:00000080
"DescriptionID"=dword:00000081
"MinimumIMPIRelease"="NONE"
"MinimumRISCRelease"="030701"
"ProductID"="NONE"
"ServerEntryPoint"="sampext.dll"

<table>
<thead>
<tr>
<th>Primary Registry key field</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>If the plug-in adds new folders to the Operations Navigator hierarchy, the value of this field should be PLUGIN. Otherwise, it should be EXT.</td>
</tr>
<tr>
<td>NLS</td>
<td>Identifies the name of the resource DLL that contains the locale-dependent resources for the plug-in. In the development version of the registry file, this may be a fully-qualified pathname.</td>
</tr>
<tr>
<td>NameID</td>
<td>A double word containing the resource identifier of the text string in the resource DLL which will be used to identify the plug-in in the Operations Navigator user interface.</td>
</tr>
<tr>
<td>DescriptionID</td>
<td>A double word that contains the resource identifier of the text string in the resource DLL. This resource DLL is used to describe the function of the plug-in in the Operations Navigator user interface.</td>
</tr>
<tr>
<td>MinimumIMPIRelease</td>
<td>A 6-character string that identifies the minimum release of OS/400 that runs on the IMPI hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 2 Modification Level 0, the value of this field should be &quot;030200.&quot; If the plug-in does not support any OS/400 release that runs on IMPI hardware (releases prior to Version 3 Release 6), the value of this field should be &quot;NONE.&quot; If the plug-in can support any release that runs on IMPI hardware, the value of this field should be &quot;ANY.&quot;</td>
</tr>
<tr>
<td>MinimumRISCRelease</td>
<td>A 6-character string that identifies the minimum release of OS/400 that runs on RISC hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 7 Modification Level 1, the value of this field should be &quot;030701.&quot; If the plug-in does not support any OS/400 release that runs on RISC hardware (Version 3 Release 6 and above), the value of this field should be &quot;NONE.&quot; If the plug-in can support any release that runs on RISC hardware, the value of this field should be &quot;ANY.&quot;</td>
</tr>
<tr>
<td><strong>ProductID</strong></td>
<td>A 7-character string that specifies the product ID of a prerequisite iSeries server licensed program that is required by the plug-in. If the plug-in does not require that a particular licensed program be installed on the iSeries server, the value of this field should be “NONE.” Multiple comma-separated product IDs may be specified if multiple IDs exist for the same product.</td>
</tr>
<tr>
<td><strong>ServerEntryPoint</strong></td>
<td>The name of the code DLL that implements the server entry point. This entry point is called by the Operations Navigator when it needs to determine whether the plug-in is supported on a particular iSeries server. If the plug-in does not implement the entry point, the value of this field should be “NONE.” In the development version of the registry file, this may be a fully-qualified pathname.</td>
</tr>
<tr>
<td><strong>JavaPath</strong></td>
<td>The classpath string that identifies the location of your plug-in’s Java classes. During development of your plug-in, this field might contain the directory paths for the directories where your class files reside. In the production version of the registry file, it should identify your JAR file names relative to the Client Access install path, each preceded by the Client Access substitution variable that represents the install path.</td>
</tr>
<tr>
<td><strong>JavaMRI</strong></td>
<td>The base names of the JAR files that contain locale-dependent resources for the plug-in. Operations Navigator will search for each JAR file after first suffixing the name with the appropriate Java language and country identifiers. If no MRI JAR files exist for a given locale, Operations Navigator will expect the MRI for the base locale (usually US English) to reside in the code JAR files.</td>
</tr>
</tbody>
</table>

**Shell plug-ins**

These registry keys map a particular node or set of nodes in the hierarchy to the type of function supplied by the plug-in, and to the CLSID of the implementation class which implements the function.

Remember that any number of shell plug-ins may register their intent to add function to a given object type in the Navigator hierarchy. The plug-in should never assume that it is the only server component which is providing function for a given object type. This applies not only to existing object types, but also to any new objects that a plug-in may choose to define. If your plug-in is widely used, there is nothing to prevent another vendor from extending object types that are defined by your plug-in.

**Object type identifiers**

A pair of object type identifiers, subkeys `\Sample\^\`, are always expected at this level in the subkey hierarchy.

The first identifier in the pair specifies the root folder for a Navigator component. For plug-ins that add new folders, this identifier should always match the registry key name for a root folder specified the previous section. For plug-ins which add behaviors to existing object types, this subkey should generally be the
object type of the first-level folder under an iSeries server container object. These type strings are defined under HKEY_CLASSES_ROOT\IBM.AS400.Network\TYPES in the registry.

The second identifier in the pair identifies the specific object type that the plug-in wants to affect. If * is specified, the plug-in will be called the for the folder type identified in the parent subkey, plus all folders and objects which appear in the hierarchy under that folder. Otherwise, a specific type identifier must be specified, and the plug-in will then only be called for that object type.

Checking for object types

When performing checks for existing object types, you should use the 3-character type identifiers that are defined under the key HKEY_CLASSES_ROOT\IBM.AS400.Network\TYPES in the registry. When performing checks for new object types that are defined by your plug-in, use a registry key. Use the registry key that identifies the folder that you specified as your junction point, or whatever type you will return to the Navigator when serving data for a folder that is defined by your plug-in.

Customize the VB plug-in registry values

The sample plug-includes two registry files: VBSMPDBG.REG, a windows-readable registry file for use during development and VBSMPRLS.REG, a registry file for distribution on the iSeries server. The following table describes the sections in this registry file, and recommends changes for use when developing your own plug-in.

Primary registry key

The primary registry key defines a set of fields which specify global information for the plug-in. This information is required.

Note: The subkey name must match the ProgID for your plug-in.

See Example: Primary registry key for a description of each field.

```plaintext
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.VBSample] "Type"="Plugin" "NLS"="vbsmpmri.dll" "NameID"="vbsample.dll"
```

Recommended changes:
1. Change the name “vbsample.dll” in the ServerEntryPoint key to match the name of the plug-in ActiveX server DLL.
2. Change the name “vbsmpmri.dll” in the NLS key to match the name of the C++ MRI resource DLL for your plug-in. Each Visual Basic plug-in must have a unique C++ MRI DLL name.

Note: Do not include the path in either of these changes.

Registering a new folder

This section will register a Visual Basic Plug-in ListManager class implementation for each new folder added to the Operations Navigator hierarchy. If your plug-in does not add any new folders to the Operations Navigator hierarchy, delete this section and proceed to the next task.

The Visual Basic ListManager class is the main interface to serve data to your plug-in folder.

The sample places the Sample Visual Basic Folder into the root level of an iSeries server system name in the Operations Navigator hierarchy. If you want your folder to appear at some other point in the hierarchy, you must change the "Parent" key value. See Parent field values for a listing of possible values.
See Example: New folder registry key for a description of each field, and the possible values.

```
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.VBSample\folders\SampleVBFolder]
"Parent"="AS4"
"Attributes"=hex:00,01,00,20
"CLSID"="{040606B1-1C19-11d2-AA12-08005AD17735}"
"VBClass"="vbsample.SampleListManager"
"VBInterface"="{0FC5EC72-8E00-11D2-AA9A-08005AD17735}"
"NameID"=dword:00000082
"DescriptionID"=dword:00000083
"DefaultIconIndex"=dword:00000001
"OpenIconIndex"=dword:00000001
```

Recommended changes:

1. Change all occurrences of the name “SampleVBFolder” in the registry file to a unique name that will identify your folder object. The name that is specified in the registry file must match the object name that is specified in your ListManager and ActionsManager Visual Basic classes. For the sample plug-in these Visual Basic source files are: `listman.cls` and `actnman.cls`.

2. Change the name “vbsample.SampleListManager” in the `VBClass` key to match the program identifier name of your ListManager class. For example, if your ActiveX Server DLL is named foo.dll, and your ListManager implementation class is MyListManager, then the program identifier is “foo.MyListManager”. This name is case-sensitive.

3. Change the value of the “`VBInterface`” key to the ListManager implementation class interface ID.

**Registering VB plug-in objects**

The final section of the registry specifies which objects in the Navigator hierarchy are affected by implementation of the Visual Basic plug-in.

On many of the ActionsManager, ListManager and DropTargetManager class methods, you will be passed in items or objects. To determine which folder object is being referenced, use the object type string that is defined in the Windows registry.

Property sheets still can be added to your plug-in by using a context menu item. You cannot use a registry key for a property sheet that is the mechanism that is used for a C++ plug-in. Property sheet handlers including the Auto Refresh property sheet handler are not supported for Visual Basic plug-ins.
Recomended changes:
1. The CLSID in the entries above should always have the following: "(040606B2-1C19-11d2-AA12-08005AD17735)"
2. The "VBClass" key contains the program identifier (ProgID) of the Visual Basic implementation class.
3. The "VBInterface" key contains the Visual Basic implementation class' interface ID.
4. If your plug-in will not be a drop handler for objects, remove the drag and drop context menu handler and drop handler registry entries.
5. Rename the subkeys \SampleVBFolder\ and use a unique string to identify your folder object. This name is the object type that will be used in your Visual Basic source to identify when actions are taken on this folder in Operations Navigator.
6. In the file that you created that was based on the ActionsManager interface, edit the code that checks for the object types that are defined by the sample to reflect the name of your new folder object. The sample's ActionsManager interface is located in actnman.cls.

Global changes:

Define a unique programmatic identifier, or ProgID for your plug-in. The ProgID should match the <vendor>.<component> text string, where vendor identifies the name of the vendor who developed the plug-in, and component describes the function being provided. In the sample plug-in, the string "IBM.Sample" identifies IBM as the vendor, and "Sample" as the description of the function that is provided by the plug-in. This will be used throughout the registry file, and will name the directory where your plug-in will reside on both the iSeries server and the workstation.

Replace all instances of "IBM.VBSample" with your new [vendor].ProgID.

Note: Operations Navigator provides built-in ActiveX server DLLs that manage plug-ins written in Java and in Visual Basic. Therefore, all Java and Visual Basic plug-ins register their own respective CLSIDs. The registry files that are provided with the programming samples already contain these predefined CLSIDs.
Example: Primary registry key

The primary registry key defines a set of fields that specify global information for the plug-in. This information is required.

;--------------------------------------------------------------------
; Define the primary registry key for the plugin
; NOTE: NLS and ServerEntryPoint DLL names must not contain qualified directory paths

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-ins\IBM.Sample]
"Type"="PLUGIN"
"NLS"="sampmri.dll"
"NameID"=dword:00000080
"DescriptionID"=dword:00000081
"MinimumIMPIRelease"="NONE"
"MinimumRISCRelease"="030701"
"ProductID"="NONE"
"ServerEntryPoint"="sampext.dll"

<table>
<thead>
<tr>
<th>Primary Registry key field</th>
<th>Field Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>If the plug-in adds new folders to the Operations Navigator hierarchy, the value of this field should be PLUGIN. Otherwise, it should be EXT.</td>
</tr>
<tr>
<td>NLS</td>
<td>Identifies the name of the resource DLL that contains the locale-dependent resources for the plug-in. In the development version of the registry file, this may be a fully-qualified pathname.</td>
</tr>
<tr>
<td>NameID</td>
<td>A double word containing the resource identifier of the text string in the resource DLL which will be used to identify the plug-in in the Operations Navigator user interface.</td>
</tr>
<tr>
<td>DescriptionID</td>
<td>A double word that contains the resource identifier of the text string in the resource DLL. This resource DLL is used to describe the function of the plug-in in the Operations Navigator user interface.</td>
</tr>
<tr>
<td>MinimumIMPIRelease</td>
<td>A 6-character string that identifies the minimum release of OS/400 that runs on the IMPI hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 2 Modification Level 0, the value of this field should be &quot;030200.&quot; If the plug-in does not support any OS/400 release that runs on IMPI hardware (releases prior to Version 3 Release 6), the value of this field should be &quot;NONE.&quot; If the plug-in can support any release that runs on IMPI hardware, the value of this field should be &quot;ANY.&quot;</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MinimumRISCRelease</td>
<td>A 6-character string that identifies the minimum release of OS/400 that runs on RISC hardware that the plug-in requires. The string should be of the form vvrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 7 Modification Level 1, the value of this field should be &quot;030701.&quot; If the plug-in does not support any OS/400 release that runs on RISC hardware (Version 3 Release 6 and above), the value of this field should be &quot;NONE.&quot; If the plug-in can support any release that runs on RISC hardware, the value of this field should be &quot;ANY.&quot;</td>
</tr>
<tr>
<td>ProductID</td>
<td>A 7-character string that specifies the product ID of a prerequisite iSeries server licensed program that is required by the plug-in. If the plug-in does not require that a particular licensed program be installed on the iSeries server, the value of this field should be &quot;NONE.&quot; Multiple comma-separated product IDs may be specified if multiple IDs exist for the same product.</td>
</tr>
<tr>
<td>ServerEntryPoint</td>
<td>The name of the code DLL that implements the server entry point. This entry point is called by the Operations Navigator when it needs to determine whether the plug-in is supported on a particular iSeries server. If the plug-in does not implement the entry point, the value of this field should be &quot;NONE.&quot; In the development version of the registry file, this may be a fully-qualified pathname.</td>
</tr>
<tr>
<td>JavaPath</td>
<td>The classpath string that identifies the location of your plug-in's Java classes. During development of your plug-in, this field might contain the directory paths for the directories where your class files reside. In the production version of the registry file, it should identify your JAR file names relative to the Client Access install path, each preceded by the Client Access substitution variable that represents the install path.</td>
</tr>
<tr>
<td>JavaMRI</td>
<td>The base names of the JAR files that contain locale-dependent resources for the plug-in. Operations Navigator will search for each JAR file after first suffixing the name with the appropriate Java language and country identifiers. If no MRI JAR files exist for a given locale, Operations Navigator will expect the MRI for the base locale (usually US English) to reside in the code JAR files.</td>
</tr>
</tbody>
</table>

**Parent field values**

A three-character ID that identifies the parent of the folder to be added. One of the following IDs may be specified:

ADF    Application Development folder
**Example: New folder registry key**

A separate registry key must be defined for the root of each sub tree of objects that a plug-in chooses to add to the object hierarchy. This key contains information specific to the root folder of the sub tree.

Assign the registry key a meaningful folder name that is at least four characters in length.

```plaintext
; Register a new folder

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY\plug-ins\IBM.Sample\folders\Sample]
"Parent"="AS4"
"Attributes"=hex:00,01,00,20
"CLSID"={D09970E1-9073-11d0-82BD-08005AA74F5C}
"NameID"=dword:00000082
"DescriptionID"=dword:00000083
"DefaultIconIndex"=dword:00000000
"OpenIconIndex"=dword:00000001
"AdminItem"="QIBM_SAMPLE_SMPFLR"
```

<table>
<thead>
<tr>
<th><strong>Parent</strong></th>
<th>A three-character ID that identifies the parent of the folder to be added. See Parent field values for a listing of possible values.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Attributes</strong></td>
<td>A 4-byte binary field that contains the attributes for the folder, with the indicator bytes in reverse order. See the folder attribute flags defined for the IShellFolder::GetAttributesOf method in the Microsoft include file SHLOBJ.H.</td>
</tr>
<tr>
<td><strong>CLSID</strong></td>
<td>The CLSID of the IA4HierarchyFolder implementation that should be called by the Operations Navigator to obtain the contents of the folder. For <strong>Java plug-ins</strong>, the CLSID always should be: 1827A856-9C20-11d1-96C3-00062912C9B2. For <strong>Visual Basic plug-ins</strong>, the CLSID should always be: 040606B1-1C19-11d2-AA12-08005AD17735.</td>
</tr>
<tr>
<td><strong>JavaClass</strong></td>
<td>The fully-qualified Java class name of the ListManager implementation that should be called by the Operations Navigator to obtain the contents of the folder. This field should be omitted if the plug-in is not a Java plug-in.</td>
</tr>
<tr>
<td><strong>VBClass</strong></td>
<td>The Program Identifier (ProgID) of the ListManager implementation class that should be called by Operations Navigator to obtain the contents of the folder.</td>
</tr>
<tr>
<td><strong>VBIface</strong></td>
<td>The GUID of the ListManager implementation class’ interface.</td>
</tr>
<tr>
<td>NameID</td>
<td>A double word that contains the resource ID of the string that should appear as the name of the folder in the Operations Navigator hierarchy.</td>
</tr>
<tr>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>DescriptionID</td>
<td>A double word that contains the resource ID of the string that should appear as the description of the folder in the Operations Navigator hierarchy.</td>
</tr>
<tr>
<td>DefaultIconIndex</td>
<td>A double word that contains the index into the NLS resource DLL of the plug-in for the icon that should be displayed for the folder in the Operations Navigator hierarchy. This is a zero-based index into the resource DLL, not the resource ID of the icon. For indexing to work properly, the icon resource IDs should be assigned sequentially.</td>
</tr>
<tr>
<td>OpenIconIndex</td>
<td>A double word that contains the index into the NLS resource DLL of the plug-in for the icon that should be displayed for the folder in the Operations Navigator hierarchy whenever it is selected by the user.</td>
</tr>
<tr>
<td>AdminItem</td>
<td>A STRING that contains the Function ID of the Application Administration function that controls access to the folder. If this field is omitted, no Application Administration function controls access to the folder. If specified, this must be the function ID of a Group or Administrable function. It cannot be the function ID of a Product Function.</td>
</tr>
</tbody>
</table>

### Sample Java registry file

Each of the sample plug-ins written in Java provides its own registry file. The following sections describe the important parts of the registry file and illustrate how to create appropriate entries for your own plug-ins. The examples are taken from the appropriate sample which illustrates the function described.

**Programmatic Identifier (ProgID)**

Your plug-in is uniquely identified to Operations Navigator by means of a text string of the form `<vendor>.<component>`, where vendor identifies the vendor who developed the plug-in, and component describes the function being provided. In the examples below, the string `IBM.MsgQueueSample3` identifies IBM as the vendor, and "MsgQueueSample3" as the description of the function provided by the plug-in. This string is known as the programmatic identifier, or ProgID. It’s used throughout the registry file when specifying the function your plug-in provides, and it also names the directory where your plug-in will reside on both the iSeries server and the client workstation.

**Globally unique identifiers (GUIDs)**

Microsoft’s Component Object Model uses 16-byte hex integers to uniquely identify ActiveX implementation classes and interfaces. These integers are known as Globally Unique Identifiers, or GUIDs. GUIDs that identify implementation classes are called CLSIDs (pronounced “class IDs”).

For Operations Navigator components written in Java, you should not define new GUIDs. All Java plug-ins use a set of standard GUIDs that specify the built-in ActiveX server component which manages Java plug-ins. The standard CLSIDs to use are provided in the examples below.
Defining your plug-in’s primary attributes

; Define the primary registry key for Message Queue Sample 3.

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.MsgQueueSample3]
"Type"="PLUGIN"
"NLS"="MessageQueuesMRI.dll"
"NameID"=dword:00000001
"DescriptionID"=dword:00000002
"MinimumIMPIRelease"="NONE"
"MinimumRISCRelease"="ANY"
"ProductID"="NONE"
"ServerEntryPoint"="NONE"
"JavaPath"="MsgQueueSample3.jar"
"JavaMRI"="MsgQueueSample3MRI.jar"

Type
If the plug-in adds new folders to the Operations Navigator hierarchy, the value of this field should be PLUGIN. Otherwise, it should be EXT.

NLS
Identifies the name of the resource DLL that contains locale-dependent resources for the plug-in. In the development version of the registry file, this may be a fully-qualified pathname.

NameID
A double word containing the resource identifier of the text string in the resource DLL which will be used to identify the plug-in in the Operations Navigator user interface.

DescriptionID
A double word that contains the resource identifier of the text string in the resource DLL. This resource DLL is used to describe the function of the plug-in in the Operations Navigator user interface.

MinimumIMPIRelease
A 6-character string that identifies the minimum release of OS/400 running on IMPI hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 2 Modification Level 0, the value of this field should be "030200."

If the plug-in does not support any OS/400 release that runs on IMPI hardware (releases prior to Version 3 Release 6), the value of this field should be "NONE." If the plug-in can support any release that runs on IMPI hardware, the value of this field should be "ANY."

MinimumRISCRelease
A 6-character string that identifies the minimum release of OS/400 running on RISC hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 7 Modification Level 1, the value of this field should be "030701."

If the plug-in does not support any OS/400 release that runs on RISC hardware (Version 3 Release 6 and above), the value of this field should be "NONE." If the plug-in can support any release that runs on RISC hardware, the value of this field should be "ANY."

ProductID
A 7-character string that specifies the product ID of a prerequisite iSeries server licensed program that is required by the plug-in. If the plug-in does not require that a particular licensed program be installed on the iSeries server, the value of this field should be "NONE."
Multiple comma-separated product IDs may be specified if multiple IDs exist for the same product.

**ServerEntryPoint**
The name of the code DLL that implements the server entry point. This entry point is called by the Operations Navigator when it needs to determine whether the plug-in is supported on a particular iSeries server. If the plug-in does not implement the entry point, the value of this field should be "NONE." In the development version of the registry file, this may be a fully-qualified pathname.

**JavaPath**
The classpath string that identifies the location of your plug-in’s Java classes. During development of your plug-in, this field might contain the directory paths for the directories where your class files reside. In the production version of the registry file, it should identify your JAR files. The JAR file names should not be qualified with any directory names - Operations Navigator will qualify them automatically when it constructs the classpath string to be passed to the Java VM.

**JavaMRI**
The base names of the JAR files that contain locale-dependent resources for the plug-in. Operations Navigator will search for each JAR file after first suffixing the name with the appropriate Java language and country identifiers. In the development version of the registry file this field may contain an empty string, since the resources for the base locale (usually US English) should reside in the code JAR.

### Defining new folders

```plaintext
;--------------------------------------------------------------------
; Register a new folder

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.MsgQueueSample3\folders\Sample3]
"Parent"="AS4"
"Attributes"=hex:00,01,00,a0
"CLSID"="{1827A856-9C20-11d1-96C3-00062912C9B2}"
"JavaClass"="com.ibm.as400.opnav.MsgQueueSample3.MqListManager"
"NameID"=dword:0000000b
"DescriptionID"=dword:0000000c
"DefaultIconIndex"=dword:00000001
"OpenIconIndex"=dword:00000000
"AdminItem"="QIBM_SAMPLE_SMPFLR"
"TaskpadNameID"=dword:00000003
"TaskpadDescriptionID"=dword:00000004
```

**Type**
Each new folder that your plug-in adds to the Operations Navigator hierarchy has a unique logical type. In the example above, the string Sample3 is the type which will be used to identify the currently selected folder when control is passed to your plug-in at runtime.

**Parent**
A three-character ID that identifies the parent of the folder to be added. One of the following IDs may be specified:

- **ADF** Application Development folder
- **AS4** iSeries server folder
- **BKF** Backup folder
- **BOF** Basic Operations folder
- **CFG** Configuration and Service folder
Attributes
A 4-byte binary field that contains the attributes for the folder, with the indicator bytes in reverse order. See the folder attribute flags defined for the IShellFolder::GetAttributesOf method in the Microsoft include file SHLOBJ.H. To indicate that your folder has a taskpad, use 0x00000008.

CLSID
The CLSID of the IA4HierarchyFolder implementation that should be called by Operations Navigator to obtain the contents of the folder. For Java plug-ins this CLSID should always be \{1827A856-9C20-11d1-96C3-00062912C9B2\}.

JavaClass
The fully-qualified Java class name of the ListManager implementation that should be called by the Operations Navigator to obtain the contents of the folder.

NameID
A double word that contains the resource ID of the string that should appear as the name of the folder in the Operations Navigator hierarchy.

DescriptionID
A double word that contains the resource ID of the string that should appear as the description of the folder in the Operations Navigator hierarchy.

DefaultIconIndex
A double word that contains the index into the NLS resource DLL of the plug-in for the icon that should be displayed for the folder in the Operations Navigator hierarchy. This is a zero-based index into the resource DLL, not the resource ID of the icon. For indexing to work properly, the icon resource IDs should be assigned sequentially.

OpenIconIndex
A double word that contains the index into the NLS resource DLL of the plug-in for the icon that should be displayed for the folder in the Operations Navigator hierarchy whenever it is selected by the user. This may be the same as the default icon index.

AdminItem
A STRING that contains the Function ID of the Application Administration function that controls access to the folder. If this field is omitted, no Application Administration function controls access to the folder. If specified, this must be the function ID of a Group or Administrable function. It cannot be the function ID of a Product Function.

TaskpadNameID
A double word that contains the resource ID of the string that should appear as the name of the taskpad in the Operations Navigator hierarchy.
TaskpadDescriptionID
A double word that contains the resource identifier of the text string in the resource DLL. This resource DLL is used to describe the function of the taskpad in the Operations Navigator user interface.

Adding context menu items

; Register a context menu handler for the new folder and its objects

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.MsgQueueSample3\shellex\Sample3\ContextMenuHandlers\{1827A857-9C20-11d1-96C3-00062912C9B2}\]
"JavaClass"="com.ibm.as400.opnav.MsgQueueSample3.MqActionsManager"

; Register a drag/drop context menu handler for the new folder and its objects

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.MsgQueueSample3\shellex\Sample3\DragDropHandlers\{1827A857-9C20-11d1-96C3-00062912C9B2}\]
"JavaClass"="com.ibm.as400.opnav.MsgQueueSample3.MqActionsManager"

Adding taskpad tasks

; Register a task handler for the new folder and its objects

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.MsgQueueSample5\shellex\Sample5\TaskHandlers\{1827A857-9C20-11d1-96C3-00062912C9B2}\]
"JavaClass"="com.ibm.as400.opnav.MsgQueueSample5.MqTasksManager"
"JavaClassType"="TasksManager"

Supporting drag/drop

; Register a drag handler for the new folder and its objects

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.MsgQueueSample3\shellex\Sample3\DropHandler\{1827A857-9C20-11d1-96C3-00062912C9B2}\]
"JavaClass"="com.ibm.as400.opnav.MsgQueueSample3.MqDropTargetManager"

Specifying the objects to be managed

A pair of object type identifiers is required under the shellex key. The first identifier in the pair specifies the root folder for an Operations Navigator component. For new folders added by your plug-in, this identifier should match the logical type of the folder you specified as your junction point. For existing folders, this subkey should generally be the object type of the first-level folder under an iSeries server container object. These type strings are defined under HKEY_CLASSES_ROOT\IBM.AS400.Network\TYPES in the registry.

The second identifier in the pair identifies the specific object type that the plug-in wants to affect. If "*" is specified, the plug-in will be called for the folder type identified in the first identifier, plus all folders and objects which appear in the hierarchy under that folder. Otherwise, a specific type identifier should be specified, and the plug-in will only be called when the user performs an action on an object of that type.
Remember that any number of plug-ins may register their intent to add function to a given object type in the Navigator hierarchy. The plug-in should never assume that it is the only server component which is providing function for a given object type. This applies not only to existing object types, but also to any new objects that a plug-in may choose to define. If your plug-in is widely used, there is nothing to prevent another vendor from extending object types that are defined by your plug-in.

**CLSIDs**
The CLSIDs shown in the above examples specify the built-in ActiveX server component which manages Java plug-ins. For all non-folder related function this CLSID should always be `{1827A857-9C20-11d1-96C3-00062912C9B2}`.

**JavaClass**
The fully-qualified Java class name of the interface implementation that should be called by the Operations Navigator to support the designated function.

**SSL support**
If a plug-in’s communications with the iSeries server are performed by using the Sockets API or some other low-level communications service, then it is the responsibility of the plug-in to support SSL if it has been requested. If the plug-in doesn’t provide this support, it should indicate that it doesn’t support SSL as described below. When this is done, the plug-in’s function will be disabled if the user has requested a secure connection.

```plaintext
;---------------------------------------------------------------
; Indicate that this plug-in supports SSL.
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.MsgQueueSample3\SSL]
"Support Level"=dword:00000001
```

**Support Level**
If the plug-in supports SSL, this value should be 1. Otherwise, it should be 0.

**Property pages for a property sheet handler**
The Microsoft Foundation Class Library classes cannot be used to construct property pages for a property sheet handler. However, IBM provides `CExtPropertyPage`, which may be used in place of the MFC class `CPropertyPage`. Property pages implemented by Operations Navigator plug-ins should subclass `CExtPropertyPage`. The class declaration may be found in the header file PROPEXT.H, and the implementation is contained in the file PROPEXT.CPP. Both files are provided as part of the sample plug-in.

**Note** It is necessary to include PROPEXT.CPP in the project workspace for your plug-in.

If a plug-in requires that a property sheet is associated with one of its own object types, the SFGAO_HASPROPSHEET flag must be returned as part of the attributes of the object. When this flag is on, the Navigator automatically will add Properties to the context menu for the object. Also, when this flag is on, Navigator will call any registered property sheet handlers to add pages to the property sheet when the context menu item is selected.

In certain cases a plug-in may desire to implement a Properties context menu item that is defined for one of its own object types as a standard Windows dialog instead of a property sheet. A flag is defined for this situation that may be returned to the Navigator on calls to IContextMenu::QueryContextMenu. If the flag is returned, no automatic processing for Properties is performed, and it is up to the plug-in to add the context menu item and implement the associated dialog. This flag is documented in Description of QueryContextMenu flags.
If a plug-in intends to add property pages to one of the property sheets for an iSeries user, the key that specifies the CLSID of the property sheet handler must specify a PropSheet field that identifies the property sheet to which the specified handler will add pages. An example follows.

```
;--------------------------------------------------------------------
Register a property sheet handler for the Network property sheet for AS/400 users
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-ins\IBM.Sample\shellex\Users and Groups\User\PropertySheetHandlers\{3D7907A1-9080-11d0-82BD-08005AA74F5C}]
"PropSheet"="Networks"
```

Valid values for the PropSheet field are:

<table>
<thead>
<tr>
<th>PropSheet field valid values</th>
<th>Groups</th>
<th>Personal</th>
<th>Security or Capabilities</th>
<th>Jobs</th>
<th>Networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groups-Before-All</td>
<td>Personal-Before-All</td>
<td>Jobs-Before-All</td>
<td>Networks-Before-All</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Groups-After-Info</td>
<td>Personal-After-Privileges</td>
<td>Jobs-After-General</td>
<td>Networks-After-General</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Personal-After-Auditing</td>
<td>Jobs-After-Startup</td>
<td>Networks-After-Startup</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Personal-After-Other</td>
<td>Jobs-After-Display</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Capabilities-After-Other</td>
<td>Jobs-After-Output</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Jobs-After-International</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

To add pages to a property sheet for an iSeries user, the plug-in must implement the IA4PropSheetNotify interface (see IA4PropSheetNotify interface specifications listing).

**Restriction:**

The following restriction currently applies to property sheets for iSeries user objects:

Multiple property sheet handlers for the various property sheets that are associated with an iSeries user cannot be implemented on the same implementation class. Each property sheet requires a separate CLSID.

**Description of QueryContextMenu flags**

Operations Navigator supports the following enhancements to the IContextMenu interface:

**Ordering of context menu items**

The Operations Navigator has extended the IContextMenu interface to obtain more precise control over the order in which menu items are added to the menu for a particular folder or object. The Navigator structures its context menus in three sections. This structure ensures that when more than one component adds items to the context menu for an object, the items will still appear in the correct order that is defined for the Windows user interface.
The first section contains actions which are specific to the object type, such as Reorganize for a database table. The second section contains "object creation" items; these items are object types which cascade off of a New menu item. Lastly there are the so-called "standard" Windows menu items, such as Delete or Properties. You may choose to add menu items to any section of the context menu.

The Operations Navigator calls the QueryContextMenu method for a component three times in succession, once for each section of the menu. The following additional flags are defined in the uFlags parameter to allow you to determine which section of the context menu is currently being serviced.

**UNITY_CMF_CUSTOM**
This flag indicates that you should add object-specific actions to the menu.

**UNITY_CMF_NEW**
This flag indicates that you should add object creation items to the menu.

**UNITY_CMF_STANDARD**
This flag indicates that you should add standard actions to the menu.

**UNITY_CMF_FILEMENU**
This flag changes UNITY_CMF_STANDARD. It indicates construction of the File menu pull down for your object, as opposed to the menu that is displayed when the user clicks on an object with mouse button 2.

Items on the File pull down are arranged slightly differently. If you add Properties to the menu, you should avoid inserting a separator as is normally done before this item. Also, edit actions such as Copy or Paste should not be added to the File menu, because they appear on the Edit pull down instead. (The Operations Navigator calls your shell plug-in at the appropriate time to obtain the items for the Edit menu, and does not set UNITY_CMF_FILEMENU).

**Unique property dialogs**

In certain cases, a plug-in may desire to implement a Properties context menu item that is defined for one of its own object types as a standard Windows dialog instead of a property sheet. A flag that is defined for this situation may be returned to the Navigator on calls to IContextMenu::QueryContextMenu when the UNITY_CMF_STANDARD flag is set. This flag, A4HYF_INFO_PROPERTIESADDED, should be OR’d with the HRESULT value that is returned by QueryContextMenu.

Returning this flag means that automatic processing for Properties is not performed. In this case, the plug-in must add the context menu item and construct the associated dialog.

**Example: Constructing Visual Basic property pages for a property sheet handler**

Property pages that are implemented by Operations Navigator Visual Basic plug-ins can not use a registry key to specify property pages. You must add a specific property page context menu item in your ListManager class to implement a property page. You can not add a property page to any existing property sheet objects.

In the Visual Basic Sample plug-in, a property page is supported for Libraries in the Operations Navigator List. This is done with the following steps:

1. In listman.cls, the Library object type specifies a properties page in the getAttributes method:

   `Public Function ListManager_getAttributes(ByVal item As Object) As Long
   Dim uItem As ItemIdentifier
   Dim nAttributes As ObjectTypeConstants
   ' Returns the attributes of an object in the list.
   nAttributes = (uItem.Type = 104) ' iSeries: Developing Operations Navigator plug-ins
   Return nAttributes
   End Function`
If Not IsEmpty(item) Then
    Set uItem = item
End If

If uItem.getType = "SampleVBFolder" Then
    nAttributes = OBJECT_ISCONTAINER
ElseIf item.getType = "SampleLibrary" Then
    nAttributes = OBJECT_IMPLEMENTSPROPERTIES
Else
    nAttributes = 0
End If

ListManager_getAttributes = nAttributes
End Function

2. In actnman.cls, the queryActions method specifies that properties should be shown on the Library object context menu.

Public Function ActionsManager_queryActions(ByVal flags As Long) As Variant
    ' Add menu items to a Sample Library
    If selectedFolderType = "SampleLibrary" Then
        ' Standard Actions
        If (flags And STANDARD_ACTIONS) = STANDARD_ACTIONS Then
            ReDim actions(0)

            ' Properties
            Set actions(0) = New ActionDescriptor
            With actions(0)
                .Create
                .setID IDPROPERTIES
                .SetText m_uLoader.getString(IDS_ACTIONTEXT_PROPERTIES)
                .setHelpText m_uLoader.getString(IDS_ACTIONHELP_PROPERTIES)
                .setVerb "PROPERTIES"
                .setEnabled True
                .setDefault True
            End With

            ' Properties is only selectable if there is ONLY 1 object selected
            If Not IsEmpty(m_ObjectNames) Then
                If UBound(m_ObjectNames) > 0 Then
                    actions(2).setEnabled False
                End If
            End If
        End If
    End If
End Function

3. In actnman.cls, the actionsSelected method displays a properties form when the properties context menu is selected.

Public Sub ActionsManager_actionSelected(ByVal action As Integer, ByVal owner As Long)
    Select Case action
        Case IDPROPERTIES
            If (Not IsEmpty(m_ObjectNames)) Then
                ' Pass the System Name into a hidden field on the form for later use
                frmProperties.lblSystemName = m_ObjectNames(0).getSystemName

                ' Pass the Display Name of the selected object into a hidden field on the form
                frmProperties.lblLibName = m_ObjectNames(0).getDisplayName
            End If
    End Case
End Sub
Note: The code to create and display the property sheet can

Secure Sockets Layer (SSL) registry entry

Operations Navigator users can request a secure connection to an iSeries server by selecting the Use Secure Sockets Layer checkbox on the Connection tab of the property sheet for iSeries objects. When this is done, only Operations Navigator components that are capable of supporting SSL communications are enabled for activation by the user.

If all of a plug-in’s communications with the iSeries server are managed by using the Client Access system handle (enter cwbCO_SysHandle), or by using the class com.ibm.as400.access.AS400 in the case of a Java plug-in, then it should indicate that it supports secure connections to the iSeries server. For C++ plug-ins, the cwbCO_SysHandle is obtained by calling the cwbUN_GetSystemHandle API. When the user requests a secure connection, the Navigator automatically will enable SSL. In the case of Java plug-ins, the iSeries server object obtained by calling the getSystemObject method on the class com.ibm.as400.opnav.ObjectName actually will be an instance of com.ibm.as400.access.SecureAS400.

Note: If you are running Java over SSL, and creating your own CA certificate, Client Access GA service pack is required.

If a plug-in’s communications with the iSeries server are performed by using the Sockets API or some other low-level communications service, then it is the responsibility of the plug-in to support SSL if it has been requested. If the plug-in doesn’t provide this support, it should indicate that it doesn’t support SSL as described below. When this is done, the plug-in’s function will be disabled if the user has requested a secure connection.

Example: Adding a registry key to enable SSL

The key is SSL under [HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.Sample\SSL] "Support Level"=dword:00000001 where IBM.Sample is the plug-in supplied product component.

Note: "Support Level"=dword:00000001 = supports SSL, and "Support Level"=dword:00000000 = does NOT support SSL.
Property pages for a property sheet handler

In V5R1, you can add property pages to property sheets of Java plug-ins. This allows you to build object
names, display properties, share objects with third parties, and mix C++ and Java code in the same
plug-in.

To use property pages, you must build the properties manager interface, which provides the following
methods:

- Initialize
  Identifies the container object for the properties.
- getPages
  Construct and provide a vector of PanelManager objects.
- CommitHandlers
  Returns a vector of handlers to be called upon Commit.
- CancelHandlers
  Returns a vector of handlers to be called upon Cancel.

Then enable the properties menu by having the ListManager getAttributes method return
ListManager.OBJECT_HASPROPERTIES.

Finally, create a registry entry that identifies the PropertiesManagerInterface. For example:

[HKEY_CLASSES_ROOT\IBM.AS400.Network\AS/400 Network\*
 \shell\ext\PropertySheetHandlers\{1827A857-9C20-11d1-96C3-00062912C9B2}\]
"JavaClass"="com.ibm.as400.opnav.TestPages.TestPropertiesManager"
"JavaClassType"="PropertiesManager"

Note: Multiple PropertiesManager implementations may register to provide property pages for a given object
type. Do not assume that your entity is the only one supplying pages, or the order that the pages will
be added.

For more information, see the Properties Manager example.

Example: Java Properties Manager
package com.ibm.as400.opnav.Sample;

import com.ibm.as400.opnav.*;
import java.awt.Frame;
import com.ibm.as400.ui.framework.java.*;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class SamplePropertiesManager implements PropertiesManager {
    // The list of selected objects.
    ObjectName[] m_objectNames;
// Save the array of selected object names
//
public void initialize(ObjectName[] objectNames)
{
    m_objectNames = objectNames;
}

// Return an array of Panel Managers
//
public PanelManager[] getPages()
{
    // Instantiate the data beans
    MyDataBean dataBean = new MyDataBean();
dataBean.load();
    AnotherDataBean dataBean2 = new AnotherDataBean();
dataBean2.load();

    DataBean[] dataBeans = { dataBean };
    DataBean[] dataBeans2 = { dataBean2 };

    // Create the panel
    PanelManager pm = null;
    PanelManager pm2 = null; try
    {
        pm = new PanelManager("com.ibm.as400.opnav.Sample.Sample",
                                "PAGE1",
                                dataBeans);

        pm2 = new PanelManager("com.ibm.as400.opnav.Sample.Sample",
                                "PAGE2",
                                dataBeans2);
    }
    catch (com.ibm.as400.ui.framework.java.DisplayManagerException e)
    {
        Monitor.logError("SamplePropertiesManager: Exception when creating pages "+e);
    }

    pm.setTitle("First Java Page");
    pm2.setTitle("Second Java Page");

    PanelManager[] PMArray = {pm, pm2};
    return PMArray;
}

// Return a list of ActionListener objects to be notified when commit is processed
public ActionListener[] getCommitListeners()
{
    ActionListener[] al = new ActionListener[1];
al[0] = new ActionListener()
    {
        public void actionPerformed(ActionEvent evt)
        {
            Monitor.logError("SamplePropertiesManager: Processing Commit Listener");
        }
    };  
    return al;
}

// Return a list of ActionListener objects to be notified when cancel is selected
public ActionListener[] getCancelListeners()
{
    ActionListener[] al = new ActionListener[1];
al[0] = new ActionListener()
    {
        public void actionPerformed(ActionEvent evt)
        {
            Monitor.logError("SamplePropertiesManager: Processing Cancel Listener");
        }
    };  
    return al;
}