



System i

Connecting to System i

Developing System i Navigator plug-ins

Version 6 Release 1





System i

Connecting to System i

Developing System i Navigator plug-ins

Version 6 Release 1

Note

Before using this information and the product it supports, read the information in “Notices,” on page 93.

This edition applies to version 6, release 1, modification 0 of IBM i5/OS (5761-SS1) and to all subsequent releases and modifications until otherwise indicated in new editions. This version does not run on all reduced instruction set computer (RISC) models nor does it run on CISC models.

© Copyright International Business Machines Corporation 2004, 2008. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Developing System i Navigator plug-ins 1

I What's new for V6R1	1
PDF file for Developing System i Navigator plug-ins	1
Plug-in support in System i Navigator	2
What you can do with a plug-in	2
How plug-ins work	2
Plug-in requirements	4
Distributing plug-ins	5
Setup.ini file	7
Example: Information section of setup.ini	7
Example: Service section of setup.ini	8
Example: Identify files section of setup.ini	9
Example: Exit program section of setup.ini	11
MRI setup file	13
Identifying plug-ins to System i Navigator	13
Installing and running sample plug-ins	13
Setting up sample C++ plug-ins	14
Setting up sample Visual Basic plug-ins	15
Sample Visual Basic plug-in directory of files	16
Setting up the sample Java plug-ins	18
Sample Java plug-in directory of files	18
Plug-in programming reference	20
C++ reference	20
System i Navigator structure and flow of control for C++ plug-ins	20
System i Navigator COM interfaces for C++	21
Description of IA4HierarchyFolder Interface	22
IA4HierarchyFolder interface specifications listing	23
Description of IA4PropSheetNotify interface	30
IA4PropSheetNotify interface specifications listing	30
System i Navigator APIs	32
System i Navigator API listing	32
cwbUN_GetSystemValue	34
cwbUN_GetSystemHandle	35
cwbUN_ReleaseSystemHandle	36
cwbUN_CheckObjectAuthority	36
cwbUN_CheckSpecialAuthority	37
cwbUN_CheckAS400Name	37
cwbUN_GetUserAttribute	38
cwbUN_ConvertPidlToString	39
cwbUN_GetDisplayNameFromItemId	39
cwbUN_GetDisplayNameFromName	40
cwbUN_GetDisplayPathFromName	41
cwbUN_GetIndexFromItemId	41
cwbUN_GetIndexFromName	42
cwbUN_GetIndexFromPidl	42
cwbUN_GetListObject	42
cwbUN_GetParentFolderNameFromName	43
cwbUN_GetParentFolderPathFromName	43
cwbUN_GetParentFolderPidl	44
cwbUN_GetSystemNameFromName	44
cwbUN_GetSystemNameFromPidl	45

cwbUN_GetTypeFromName	46
cwbUN_GetTypeFromPidl	46
cwbUN_RefreshAll	47
cwbUN_RefreshList	47
cwbUN_RefreshListItems	47
cwbUN_UpdateStatusBar	48
cwbUN_GetODBCConnection	48
cwbUN_EndODBCConnections	49
cwbUN_GetIconIndex	49
cwbUN_GetSharedImageList	50
cwbUN_GetAdminValue	50
cwbUN_GetAdminValueEx	51
cwbUN_GetAdminCacheState	52
cwbUN_GetAdminCacheStateEx	53
cwbUN_IsSubcomponentInstalled	54
cwbUN_OpenLocalLdapServer	54
cwbUN_FreeLocalLdapServer	55
cwbUN_GetLdapSvrPort	55
cwbUN_GetLdapSvrSuffixCount	56
cwbUN_GetLdapSvrSuffixName	56
cwbUN_OpenLdapPublishing	57
cwbUN_FreeLdapPublishing	58
cwbUN_GetLdapPublishCount	58
cwbUN_GetLdapPublishType	59
cwbUN_GetLdapPublishServer	60
cwbUN_GetLdapPublishPort	60
cwbUN_GetLdapPublishParentDn	61
Return codes unique to System i Navigator APIs	62
Visual Basic reference	64
System i Navigator structure and flow of control for Visual Basic plug-ins	64
System i Navigator Visual Basic interfaces	65
System i Navigator ListManager interface class	65
System i Navigator ActionsManager interface class	66
System i Navigator DropTargetManager interface class	66
Java reference	66
System i Navigator structure and flow of control for Java plug-ins	66
Customizing the plug-in registry files	67
Customizing the C++ registry values	68
Primary registry key	68
Data server implementation	70
Shell plug-in implementation class	70
Shell plug-in implementation for objects	71
Global changes for C++ plug-in registry files	73
Customizing the Visual Basic plug-in registry values	73
Primary registry key	74
Visual Basic plug-in implementation class	76
Visual Basic plug-in implementation objects	78

Global changes for Visual Basic plug-in registry files	79
Sample Java registry file	79
Property pages for a property sheet handler	84
Description of QueryContextMenu flags	85
Example: Constructing Visual Basic property pages for a property sheet handler	86
Property sheet handling in Java	88

Example: Java Properties Manager.	88
Secure Sockets Layer registry entry	90

Appendix. Notices 93

Programming interface information	94
Trademarks	95
Terms and conditions	95

Developing System i Navigator plug-ins

With the plug-in feature of System i[™] Navigator, you can integrate your system administration tasks and client/server programs into a single application environment.

You can use plug-ins to consolidate third-party applications and specialized functions written in C++, Visual Basic, or Java[™] into the System i Navigator interface. Use this topic collection to help you understand what plug-ins are, how to create or customize them, and how to distribute them to your users.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 91.

What's new for V6R1



Read about new or significantly changed information for the Developing System i Navigator plug-ins topic collection.

New field in setup.ini

The information section of the setup.ini file has a new field called EclipseHelp. This field indicates whether the plug-in application uses the Eclipse platform to develop the help.

How to see what's new or changed

To help you see where technical changes have been made, the information center uses:

- The  image to mark where new or changed information begins.
- The  image to mark where new or changed information ends.

In PDF files, you might see revision bars (|) in the left margin of new and changed information.

To find other information about what's new or changed this release, see the Memo to Users.

PDF file for Developing System i Navigator plug-ins

Use this to view and print a PDF of this information.


To view or download the PDF version of this document, select Developing System i Navigator plug-ins (about 960 KB).

Saving PDF files

To save a PDF on your workstation for viewing or printing:

1. Right-click the PDF link in your browser.
2. Click the option that saves the PDF locally.
3. Navigate to the directory in which you want to save the PDF.
4. Click **Save**.

Downloading Adobe Reader

You need Adobe® Reader installed on your system to view or print these PDFs. You can download a free copy from the Adobe Web site (www.adobe.com/products/acrobat/readstep.html) .

Plug-in support in System i Navigator

System i Navigator plug-in support provides a convenient way to integrate your own functions and applications into a single user interface that is called System i Navigator.

These new functions and applications can vary in complexity from simple new behaviors to whole applications. Regardless of what specific new ability your plug-in provides, integrating it into System i Navigator provides several important benefits. For example, bundling common system tasks into a single location in System i Navigator can dramatically simplify common administration and operation functions. Also, System i Navigator's graphical interface ensures that your integrated functions can be completed easily, and with only minimal prerequisite skills.

What you can do with a plug-in

Plug-ins are sets of predefined classes and methods that System i Navigator starts in response to a particular user action.

You can use plug-ins to add or modify objects and folders in the System i Navigator hierarchy that represent your tools and applications. You can completely customize the support for your folders and objects by adding or modifying the following items:

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, for example 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 System i Navigator tree hierarchy.

How plug-ins work

After identifying the new plug-in to the Windows® registry, System i Navigator finds the new plug-in and installs it in a new configuration. Afterward, the new container appears in the System i Navigator hierarchy. When the user selects the container, the plug-in's code is called to obtain the container's contents.

System i Navigator communicates with the plug-in by calling methods defined on the ListManager interface. This interface enables applications to supply list data to the System i Navigator tree and to list views. To integrate your application into System i Navigator, you create a new class that implements this interface. The methods on the new class call your existing application to obtain the list data.

Figure 1 on page 3 shows how a Java plug-in that adds a new container to the System i Navigator tree can work.

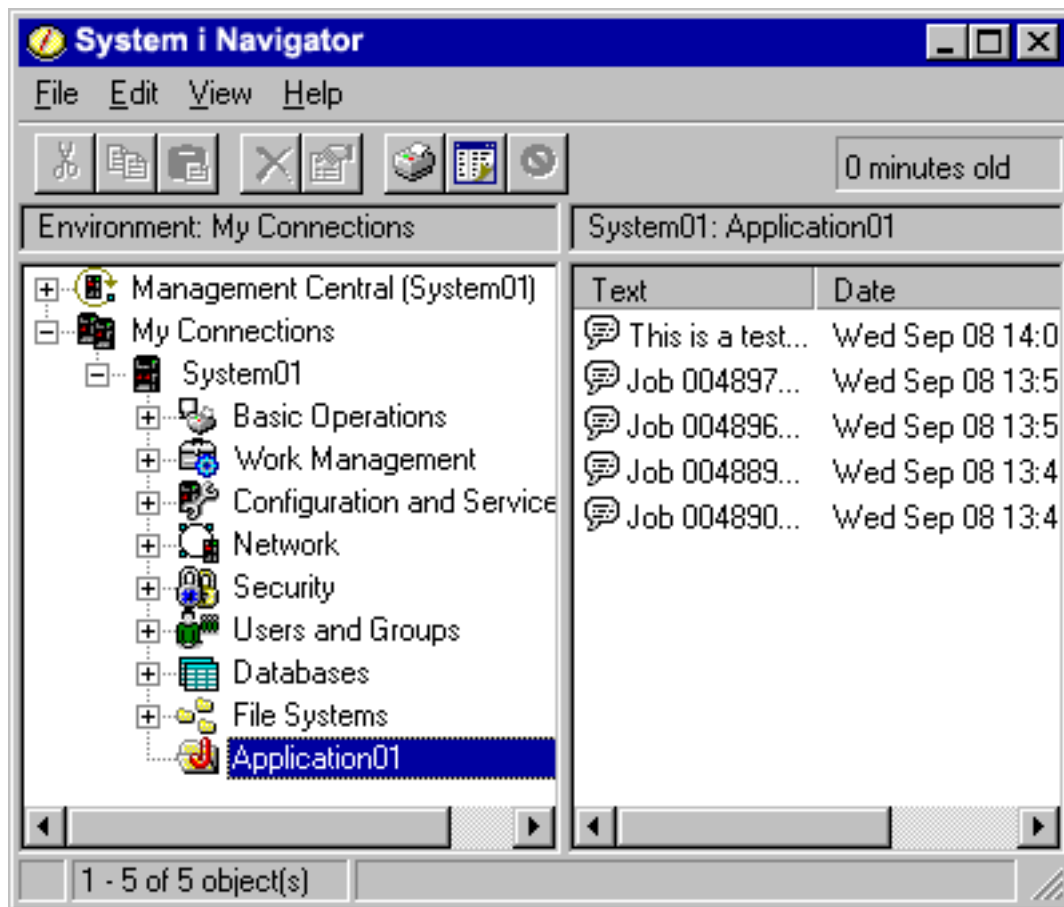


Figure 1. System i Navigator dialog that shows messages in the message queue

Figure 2 shows how System i Navigator communicates with the Java plug-in to obtain the list data.

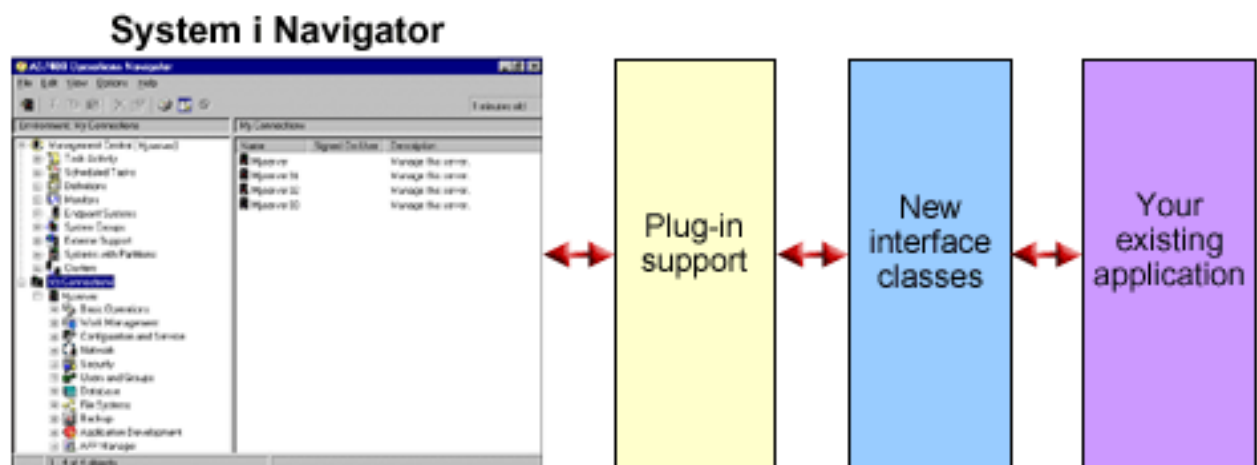


Figure 2. How System i Navigator calls an application to obtain list data

Use the ActionsManager Java interface to make your application's specialized functions available to your users through System i Navigator. When a user selects a menu item, System i Navigator calls another ActionsManager method to perform the action (you need to create a new Java class that implements this

interface). Your ActionsManager implementation calls your existing Java application, which then displays a confirmation dialog or some other more complex user interface panel that helps the user perform a specialized task.

Figure 3 shows what happens when a user right-clicks a message object to display its context menu.

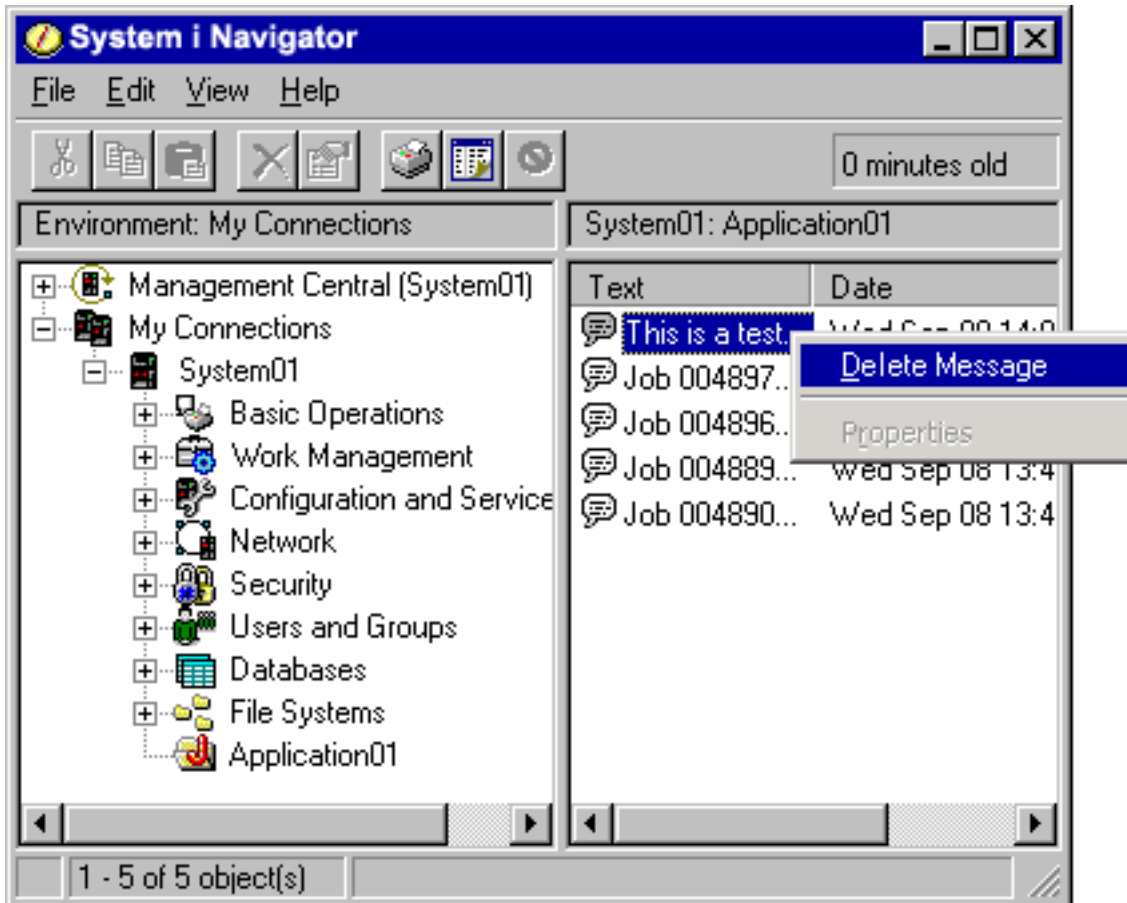


Figure 3. System i Navigator object context menu

When a user selects a menu item, System i Navigator calls another ActionsManager method to perform the action. System i Navigator calls a predefined method on the ActionsManager Java interface. This interface obtains the list of menu items supported for message objects. The System i Navigator user interface is designed to help users work with system resources. 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 by defining actions on those objects. A third interface, DropTargetManager, handles drag operations.

Plug-in requirements

System i Navigator plug-in requirements differ according to the programming language that you use.

C++ plug-ins

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

C++ plug-ins also require the following System i Navigator APIs.

Header file	Import library	Dynamic link library
cwbun.h	cwbunapi.lib	cwbunapi.dll
cwbunpla.h (Application Administration APIs)	cwbapi.lib	cwbunpla.dll

Java plug-ins

Java plug-ins run on the IBM® runtime environment for Windows, Java Technology Edition. The following table indicates the version of Java installed with the System i Access for Windows licensed program.

Release	JRE	Swing	JavaHelp
V6R1	5.0	N/A	1.1.1
V5R4	1.4.2	N/A	1.1.1
V5R3	1.4.1	N/A	1.1.1
V5R2	1.3.1	N/A	1.1.1

All Java plug-ins require a small Windows resource DLL that contains information about your plug-in. This allows System i Navigator to represent your function in the System i 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 you can use any C compiler that supports compiling and linking Windows resources.

System i 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 System i 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, found in the System i Access for Windows Toolkit.

The sample's user interface was developed by using the Graphical Toolbox for Java, which is a part of the IBM Toolbox for Java component. The Toolbox is an optionally installable feature of System i Access for Windows. It can be installed with the initial installation of the System i Access for Windows product, or selectively installed later by using the Add or Remove Programs in the Control Panel for System i Access for Windows.

Visual Basic plug-ins

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

Related concepts

"Installing and running sample plug-ins" on page 13

The Programmer's Toolkit supplies sample plug-ins in each of the supported programming languages.

Distributing plug-ins

You can deliver your plug-in code to System i Navigator users by including the code with your i5/OS® applications.

The application's installation program writes the plug-in's code binaries, registry file, and translatable resources to a folder in the integrated file system. After completing this process, your users can install the plug-in by right-clicking **My Connections** → **Install Options** → **Install Plug-ins**. The Install Plug-ins

| wizard copies your plug-in code to the users' workstations, downloads the appropriate translatable
 | resources, based on the language settings on the users' workstations, and runs the registry file to write
 | your plug-in's registry information to the Windows registry. If System i Access for Windows has not
 | already been installed, your users can install plug-ins during the initial installation using the custom
 | setup type.

For this type of plug-in	Install in this directory	And include these files
C++	/QIBM/USERDATA/OpNavPlugin/ <vendor>.<component>	<ul style="list-style-type: none"> • The registry file for the plug-in. • The System i Access for Windows setup file for the plug-in. • The ActiveX server DLL for the plug-in, and any associated code DLLs.
Java	/QIBM/USERDATA/OpNavPlugin/ <vendor>.<component>	<ul style="list-style-type: none"> • The registry file for the plug-in. • The System i Access for Windows setup file for the plug-in. • The Java JAR file, which contains all Java classes, AUIML, HTML, GIF, PDML, PCML, and serialization files.
Visual Basic	/QIBM/USERDATA/OpNavPlugin/ <vendor>.<component>	<ul style="list-style-type: none"> • The registry file for the plug-in. • The System i Access for Windows setup file for the plug-in. • The ActiveX server DLL for the plug-in, and any associated code DLLs.

Notes:

- The <vendor>.<component> subdirectory must match the one specified in the registry file.
- System i Navigator does not provide support for the GUIPlugin location. You need to migrate your plug-ins from the GUIPlugin location to the OpNavPlugin location.

Additionally, all plug-ins must create at least one directory below the <vendor>.<component> subdirectory called MRI29XX, where XX identifies a supported language; for example, MRI2924 (English). 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 users have installed your plug-in, you can choose either to upgrade it at a later date or to ship bug
 | fixes. After the code is upgraded on the system, users can manually launch plug-in updates by using the
 | **Update or Service Plug-ins** option of System i Navigator.

System i Access for Windows provides uninstallation support, so your users can completely remove the plug-in from their workstations anytime. Users can learn what plug-ins are installed on their workstations by clicking the **Plug-ins** tab on the System i Navigator Properties page for the system.

Restricting access to the plug-in with Application Administration

| You can use the system-based Application Administration support in System i Navigator to control which
 | users and user groups can access your plug-in.

Setup.ini file

Your plug-in's setup.ini file provides the installation wizard with the information that is needed to install a System i Navigator plug-in on a client workstation. It also provides information that allows the Check Service Level program to determine when the plug-in needs to be upgraded or serviced.

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

The format of the file conforms to that of a standard Windows configuration (.INI) file. The file is divided into four parts:

- Plug-in information
- Service
- Sections to identify the files to install on the client workstation
- Sections to identify exit programs to run on the client workstation

Related concepts

"Sample Visual Basic plug-in directory of files" on page 16

These tables describe all of the files included with the sample Visual Basic plug-in.

Example: Information section of setup.ini:

The first section of the setup.ini file, Plug-in Info, contains global information about the plug-in.

```
[Plugin Info]
EclipseHelp=YES
ExpressMaxRelease=V6R1M0
ExpressMinRelease=V5R2M0
Name=Sample plug-in
NameDLL=sampmri.dll
NameResID=128
Description=Sample plug-in description
DescriptionDLL=sampmri.dll
DescriptionResID=129
Version=0
VendorID=IBM.Sample
JavaPlugin=YES
```

Field in [Plugin Info] section of Setup.ini	Description of field
Name	English name of the plug-in. This name is displayed during installation of the plug-in when the translated name cannot be determined.
NameDLL	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.
NameResID	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.
Description	English description of the plug-in. This description is displayed during installation of the plug-in when the translated description cannot be determined.
DescriptionDLL	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.
DescriptionResID	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.

Field in [Plugin Info] section of Setup.ini	Description of field
Version	<p>A numeric value that indicates the release level of the plug-in. This value is used to determine whether the plug-in needs to be upgraded on the client workstation. This value is incremented by some amount for each new release of the plug-in.</p> <p>The Version value is compared with the current Version value of the plug-in installed on the client workstation. When this Version value is greater than the one that already exists on the client workstation, the plug-in is updated to the new version.</p>
VendorID	The <vendor>.<component> string that is used to identify the plug-in. This string is used to create the registry key for the plug-in in the System i Access for Windows registry tree. The VendorID must be identical to the <vendor>.<component> portion of the path where the plug-in will be installed on the system.
JavaPlugin	A field that indicates whether this is a Java plug-in. For Java plug-ins, all JAR files must be installed into the \PLUGINS\<vendor>.<component> directory, and this value is used to determine whether the installation process should do this. If it is a Java plug-in and this value is set to NO or does not exist, the plug-in cannot work after it is installed.
EclipseHelp	<p>A field that indicates whether the plug-in application uses the Eclipse platform to develop the help. The Eclipse help is used only for Java plug-ins. The help for the plug-in, if it is Eclipse-enabled, is contained in a compressed file that is specified in the setup.ini file. For each language, the compressed file is taken from the correct MRI29xx directory and is extracted to the [InstallDir]\Eclipse\Plugins directory.</p> <p>If this entry does not exist, the default value is set to NO.</p> <p>If EclipseHelp=YES is specified in the setup.ini file, an EclipseHelp section should have the following information:</p> <pre>[Eclipse] EclipseDirName=com.ibm.iSeries.help plug-in name.help.doc EclipseZipFile=superzip.zip</pre>
EclipseDirName	The directory where the help files are extracted. This directory name is only required during uninstallation because the compressed file already has the directory structure in it.
EclipseZipFile	The name of the compressed file to extract in the directory.
ExpressMinRelease	The minimum release of the operating system on which the plug-in is supported (for example, V5R2M0).
ExpressMaxRelease	The maximum release of the operating system on which the plug-in is supported (for example, V6R1M0).

Example: Service section of setup.ini:

The second section of the setup.ini file, Service, gives the Check Service Level program the information it needs to determine whether a new fix level of the plug-in should be applied to the client workstation.

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

Field in [Service] section of Setup.ini	Description of field
FixLevel	<p>A numeric value that indicates the service level of the plug-in. This value must be incremented by some amount with each service release for a particular version.</p> <p>The FixLevel value is compared with the current FixLevel value of the installed plug-in on the client's computer. If this FixLevel value is greater than that of the plug-in installed on the client workstation, the plug-in is upgraded to the new FixLevel when users select the Update or Service Plug-ins option of System i Navigator. The value must be reset to zero when a plug-in is upgraded to a new version or release level.</p>
AdditionalSize	The amount of disk space that is required to store any new or additional executable files that will be added to the plug-in during servicing. The installation uses this value to determine whether the workstation has adequate disk space for the plug-in.

Example: Identify files section of setup.ini:

This part of the setup.ini file contains the information that identifies the files 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 should be specified. Do not specify directory path names. If a file section contains no entries, the section is ignored.

Note: The Programmer's Toolkit provides a sample setup file for three different sample plug-ins: C++, Java, and Visual Basic.

Section in Setup.ini	Description
[Base Files]	Files that are copied to \PLUGINS\<vendor>.<component> under the Client Access installation directory. Normally, the ActiveX server DLL (and associated code DLLs) for the plug-in reside here.
[Shared Files]	Files that are copied to the Client Access Shared directory.
[System Files]	Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory.
[Core Files]	<p>Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory. These files are common files for more than one application.</p> <p>Each common file is associated with a number that counts the amount of the applications that use the file. When an application that uses the common file is removed, the number decrements. A common file is not removed until the last application that uses the file is uninstalled.</p> <p>Typically, these files can be redistributed.</p>
[MRI Files]	Files that are copied from the MRI directories of the plug-in on the system to the CLIENT ACCESS\MRI29XX\<vendor>.<component> directories on the workstation. This is typically where the locale-dependent resources for a plug-in reside. This includes your Resource MRI DLL name.

Section in Setup.ini	Description
[Java MRI29xx] (where 29xx is the NLV feature code for the files)	Java files that are copied from the MRI29xx directory of the plug-in on the system to the same directory to which the base files are installed. This is typically where the JAR MRI29xx resources for the plug-in reside. For each MRI29xx directory supported by the Java plug-in, a Java MRI29xx section needs to list those files. This is used only by Java plug-ins.
[Help files]	The .HLP and .CNT files that are copied from the MRI directories of the plug-in on the system to the CLIENT_ACCESS\MRI29XX\<vendor>.<component> directories on the workstation. The directory path to these files is written to HKEY_LOCAL_MACHINE\SOFTWARE\MICROSOFT\WINDOWS\HELP in the Windows registry.
[Registry files]	The Windows registry file that is associated with the plug-in.
[Dependencies]	<p>The section that defines the subcomponents that must be installed before the plug-in can be installed. AS400_Client_Access_Express is needed only if the plug-in requires other subcomponents, besides the System i Navigator base support subcomponent, to be installed.</p> <p>AS400_Client_Access_Express</p> <ul style="list-style-type: none"> The subcomponents are specified in a comma-delimited list. A single subcomponent is specified as a single number (AS400_Client_Access_Express=3). The CWBAD.H header file contains a list of constants that have the prefix CWBAD_COMP_. These constants provide the numeric values that are used in the comma-delimited list for AS400_Client_Access_Express. These CWBAD_COMP_ constants identify PC5250 font subcomponents and should not be used in the AS400_Client_Access_Express value: <pre>//5250 Display and Printer Emulator subcomponents #define CWBAD_COMP_PC5250_BASE_KOREAN (150) #define CWBAD_COMP_PC5250_PDFPDT_KOREAN (151) #define CWBAD_COMP_PC5250_BASE_SIMPCHIN (152) #define CWBAD_COMP_PC5250_PDFPDT_SIMPCHIN (153) #define CWBAD_COMP_PC5250_BASE_TRADCHIN (154) #define CWBAD_COMP_PC5250_PDFPDT_TRADCHIN (155) #define CWBAD_COMP_PC5250_BASE_STANDARD (156) #define CWBAD_COMP_PC5250_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)</pre> <p>Note: The AS400_Client_Access_Express value is used if it exists; otherwise, this section is ignored.</p>
[Service Base Files]	Files that are copied to \PLUGINS\<vendor>.<component> under the System i Access for Windows installation directory.
[Service Shared Files]	Files that are copied to the System i Access for Windows Shared directory.
[Service System Files]	Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory.

Section in Setup.ini	Description
[Service Core Files]	Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory. These files are common files for more than one application. Each common file is associated with a number that counts the amount of the applications that use the file. When an application that uses the common file is removed, the number decrements. A common file is not removed until the last application that uses the file is uninstalled. Typically, these files can be redistributed.
[Service Registry Files]	The Windows registry file that is associated with the plug-in.

Example: Exit program section of setup.ini:

The final portion of the setup.ini file contains sections that identify the programs that are to be run on the client workstation before or after an installation, upgrade, or uninstallation.

The following examples show the syntax used in these exit program sections to identify and run these programs.

Example 1: Optional program to be called before files are installed during an initial installation

```
[PreInstallProgram]
Program=whatever.exe
CmdLine=
CheckReturnCode=
Wait=
```

Field in [PreInstallProgram]	Description
Program	Required. Only the file name is used if a path is specified. The program must reside in the plug-in's <vendor>.<component> path in the installation source.
CmdLine	Optional. Whatever commands are required by the specific program.
CheckReturnCode	Optional. The default is N. If this value is set to Y, the plug-in installation does not continue if the return code is nonzero. A message is not displayed if the program returns a nonzero return code, but a message is logged in the instlog.txt file.
Wait	Optional. Wait for the program to end before continuing to run. The default is Y. If CheckReturnCode=Y, then Wait=Y is used no matter what is specified here.

Example 2: Optional program to be called after files are installed during an initial installation

```
[PostInstallProgram]
Program=whatever.exe
CmdLine=
CheckReturnCode=
Wait=
```

Field in [PostInstallProgram]	Description
Program	Required. Only the file name is used if a path is specified. The program must reside in the plug-in's <vendor>.<component> directory on the workstation.
CmdLine	Optional. Whatever commands are required by the specific program.
CheckReturnCode	Optional. The same values are supported for the PostInstallProgram section as are supported for the PreInstallProgram section. The CheckReturnCode value is logged in the instlog.txt file for reference.

Field in [PostInstallProgram]	Description
Wait	Optional. Wait for the program to end before continuing to run. The same values are supported for the PostInstallProgram section as are supported for the PreInstallProgram section.

Example 3: Optional program to be called before files are uninstalled

```
[UninstallProgram]
Program=whatever.exe
CmdLine=
CheckReturnCode=
Wait=
```

Field in [UninstallProgram]	Description
Program	Required. Only the file name is used if a path is specified. The program must reside in the plug-in's <vendor>.<component> directory on the workstation.
CmdLine	Optional. Whatever commands are required by the specific program.
CheckReturnCode	Optional. The default is N. If this value is set to Y, the plug-in uninstallation does not continue if the return code is nonzero. A message is not displayed if the return code is nonzero, but a message is logged in the instlog.txt file.
Wait	Optional. Wait for the program to end before continuing to run. The default is Y. If CheckReturnCode=Y, then Wait=Y is used no matter what is specified here.

Example 4: Optional program to be called before files are upgraded

```
[PreUpgradeProgram]
Program=whatever.exe
CmdLine=
CheckReturnCode=
Wait=
```

Field in [PreUpgradeProgram]	Description
Program	Required. Only the file name is used if a path is specified. The program must reside in the plug-in's <vendor>.<component> path in the installation source.
CmdLine	Optional. Whatever commands are required by the specific program.
CheckReturnCode	Optional. The default is N. If this value is set to Y, the plug-in installation does not continue if the return code is nonzero. A message is not displayed if the return code is nonzero, but a message is logged in the instlog.txt file.
Wait	Optional. Wait for the program to end before continuing to run. The default is Y. If CheckReturnCode=Y, then Wait=Y is used no matter what is specified here.

Example 5: Optional program to be called after files have been upgraded

```
[PostUpgradeProgram]
Program=whatever.exe
CmdLine=
CheckReturnCode=
Wait=
```

Field in [PostUpgradeProgram]	Description
Program	Required. Only the file name is used if a path is specified. The program must reside in the plug-in's <vendor>.<component> path in the installation source.

Field in [PostUpgradeProgram]	Description
CmdLine	Optional. Whatever commands are required by the specific program.
CheckReturnCode	Optional. The same values are supported for the PostUpgradeProgram section as are supported in the PreUpgradeProgram section. The CheckReturnCode value is logged in the instlog.txt file for reference.
Wait	Optional. Wait for the program to end before continuing to run. The default is Y. If CheckReturnCode=Y, then Wait=Y is used no matter what is specified here.

MRI setup file

- | The MRI setup file gives the Install Plug-ins wizard the information it needs to install the
- | locale-dependent resources that are associated with a System i Navigator plug-in on a client workstation.

You must name the MRI setup file MRISSETUP.INI. A version of this file must reside in the MRI29XX subdirectory in the System i file system 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 Install Plug-ins wizard checks the Version value of the MRI file. The MRI Version value in this file
- | must match the Version value in the SETUP.INI file of the plug-in. When these values do not match, the
- | plug-in is not listed by the Install Plug-ins wizard, and no files are copied to the client workstation. The
- | Programmer's Toolkit provides a sample MRI setup file with the sample plug-in.

Related concepts

"Sample Visual Basic plug-in directory of files" on page 16

These tables describe all of the files included with the sample Visual Basic plug-in.

Identifying plug-ins to System i Navigator

Plug-ins identify themselves to System i Navigator by supplying information to the Windows registry when the plug-in software is installed on your users' workstations.

The registry entries specify the location of the plug-in code and identify the classes that implement the special System i Navigator interfaces. You can supply additional registry information that System i Navigator uses to determine whether the plug-in's function should be activated for a particular system. For example, a plug-in might require a certain minimum release of i5/OS, or it might specify that a certain product needs to be installed on the system in order for it to function.

- | When a user selects a system in the System i Navigator hierarchy tree after installing a plug-in, System i
- | Navigator examines the system 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 system. If the plug-in's requirements are satisfied, the new function is displayed in the
- | hierarchy tree. If the requirements are not met, the plug-in's function does not appear for that system.

Installing and running sample plug-ins

The Programmer's 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 Programmer's Toolkit installed, you will
- | need to install it before working with any of the sample plug-ins. You can install the Toolkit through the
- | Add or Remove Programs in the Control Panel for System i Access for Windows.

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

Related concepts

"Plug-in requirements" on page 4

System i Navigator plug-in requirements differ according to the programming language that you use.

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 System i Navigator plug-in. You can also use the sample to verify that your Developer Studio environment is set up correctly for compiling and linking plug-in code.

To set up the sample C++ plug-in on your workstation, follow these steps.

Download the C++ plug-in	Download the executable file cppsmppq.exe. When you run the file, it extracts 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 have to modify the registry file to specify the correct location for the plug-in.
Prepare to build an ActiveX server DLL	<ol style="list-style-type: none"> 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 Programmer's Toolkit - System i 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	<ol style="list-style-type: none"> 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	<p>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.</p> <p>To build the resource DLL, complete the following steps:</p> <ol style="list-style-type: none"> 1. In Developer Studio, open the File menu, 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	<p>The SAMPDBG.REG file in the MyProject directory contains registry keys that communicate the location of the sample plug-in on your workstation to System i Navigator.</p> <p>To register the ActiveX server DLL, double-click the SAMPDBG.REG file in Windows Explorer. The registry file runs, and the entries in the file are written to the Windows registry on your workstation.</p> <p>If you specified a directory other than C:\MyProject, complete the following steps before you run the registry file:</p> <ol style="list-style-type: none"> 1. Open the SAMPDBG.REG file in Developer Studio (or a preferred text editor). 2. Replace all occurrences of C:\MyProject\ with x:\<dir>, where <i>x</i> is the drive letter where your directory resides and <dir> is the name of the directory. 3. Save the file.
Run System i Navigator in the debugger	<p>To run System i Navigator and observe the sample plug-in in action, follow these steps:</p> <ol style="list-style-type: none"> 1. In Developer Studio, open the Build menu and select Debug → Go. 2. At the prompt, type the fully qualified path to the System i Navigator executable file in the System i Access for Windows installation directory on your workstation. The path is C:\PROGRAM FILES\IBM\CLIENT ACCESS\CWBUNNAV.EXE, or something similar. 3. Click OK. The main window of the System i Navigator opens. <p>A dialog in System i Navigator prompts you to scan for the new plug-in because you have just registered a new plug-in by running the SAMPDBG.REG file. After the progress indicator finishes, click OK in the resulting dialog.</p> <p>After the System i Navigator window refreshes, a new folder (a third party sample folder) appears in the hierarchy under the system that was initially selected. You can now interact with the plug-in in System i Navigator and observe its behavior in the debugger.</p>

Related information



IBM Client Access Express Toolkit - System i Navigator Plug-ins Web page

Setting up sample Visual Basic plug-ins

The sample Visual Basic plug-in adds a folder to the System i Navigator hierarchy that provides a list of i5/OS 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 two registry files, one for use during development, and another for distribution with the retail version. See the sample Visual Basic plug-in directory of files for detailed description of all the files included with the Visual Basic plug-in.

To set up the sample Visual Basic plug-in on your workstation, follow these steps.

Download the Visual Basic plug-in	Download the executable file vbopnav.exe. When you run the file, it extracts all the files associated with the plug-in. Make a new directory, C:\VBSample, and copy all the files into it. If you create a different directory, you have to modify the registry file to specify the correct location for the plug-in.
Create the Visual Basic project	Open vbsample.vpb in Visual Basic. In the Reference dialog, select IBM System i Access for Windows ActiveX Object Library and System i Navigator Visual Basic Plug-in Support . Note: If these references do not appear in your References dialog, select Browse and look for cwbx.dll and cwbnvbi.dll in the System i Access for Windows shared directory. The IBM System i Access ActiveX Object Library contains OLE automation objects that the sample application requires to make remote command calls to the system. The System i Navigator Visual Basic plug-in support contains classes and interfaces required to create a Visual Basic plug-in directory.
Build the ActiveX server DLL	Select Make from the Visual Basic file menu to build the DLL. If it does not compile or link, locate and fix the errors, and then rebuild the DLL.
Build the resource library	<ol style="list-style-type: none"> 1. Open Microsoft® Developer Studio, open the File menu, select Open Workspace, and then select the VBSample\win32 directory. 2. In the Files of Type field, specify Makefiles (*.mak). 3. Select vbsmpmri.mak and click Open. 4. Open the Build menu and select Rebuild All to compile and link the DLL. <p>Note: You do not have to create this DLL on your own. The sample includes a resource DLL that contains the translatable text strings and other locale-dependent resources for the plug-in is included with the sample. 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.</p>
Register the plug-in	Double-click the file vbsmpdbg.reg in order to register the plug-in. If you did not use the directory C:\VBSample, edit the registry file, and replace all occurrences of "C:\\VBSample\\" with the fully qualified path to the plug-in code. You must use double back slashes in the path.
Run the plug-in in System i Navigator	From System i Navigator, expand the system that you want to scan. System i Navigator detects the changes to the registry and prompts you to scan the system to verify that it is capable of supporting the new plug-in. After completing the scan, System i Navigator displays the new plug-in in the tree hierarchy.

Related information



IBM Client Access Express Toolkit - System i Navigator Plug-ins Web page

Sample Visual Basic plug-in directory of files

These tables describe all of the files included with the sample Visual Basic plug-in.

Visual Basic project file	Description
vbsample.vbp	Visual Basic 5.0 project file

Visual Basic forms	Description
authorty.frm	Set authority form
delete.frm	Confirm delete form
propsht.frm	Property Sheet form
sysstat.frm	System status form
wizard.frm	Create new library wizard form

Visual Basic modules	Description
global.bas	Global declarations.

Visual Basic class modules	Description
actnman.cls	SampleActions Manager class
dropman.cls	Sample Drop Target Manager class
library.cls	Library class
listman.cls	Sample List Manager class

Visual Basic binary files	Description
authority.frx	Set authority form binary
delete.frx	Confirm delete form binary
propsht.frx	Property Sheet form binary
sysstat.frx	System status form binary
wizard.frx	Create new library wizard form binary
vbsample.bin	Vbsample binary

Configuration settings	Description
mrisetup.ini	Installation information for plug-in's translatable resources
setup.ini	Installation information for plug-in's executable files

Registry entries	Description
vbsmpdbg.reg	Registry file for use during development.
vbsmprls.reg	Registry file used during installation.

Files for constructing the resource DLL	Description
vbsmpmri.mak	Make File
vbsmpmri.rc	RC file
vbsmpres.h	Header file

Images	Description
compass.bmp	System i Navigator icon
lib.ico	
vbsmpflr.ico	Visual Basic Sample plug-in folder in open and closed state.
vbsmplib.ico	Visual Basic Sample plug-in library icon.

Related concepts

“MRI setup file” on page 13

The MRI setup file gives the Install Plug-ins wizard the information it needs to install the locale-dependent resources that are associated with a System i Navigator plug-in on a client workstation.

“Setup.ini file” on page 7

Your plug-in's setup.ini file provides the installation wizard with the information that is needed to install a System i Navigator plug-in on a client workstation. It also provides information that allows the Check Service Level program to determine when the plug-in needs to be upgraded or serviced.

Setting up the sample Java plug-ins

The sample Java plug-ins work with message queues in the QUSRSYS library on a given system.

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 System i user ID. The second plug-in adds support for multiple message queues. The third plug-in adds the ability to drag 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 these sample Java plug-ins on your workstation, follow these steps.

Download the sample Java plug-ins	Download the executable file jvopnav.exe. When you run this file, it extracts all of the previously mentioned files. You should allow the executable file to install the files in the default directory: jvopnav\com\ibm\as400\opnav.
Identify the plug-in to System i Navigator	<ol style="list-style-type: none">1. Edit the file MsgQueueSampleX.reg in jvopnav\com\ibm\as400\opnav\MsgQueueSampleX. (X=1, 2 or 3, depending on which sample you are installing.)2. Find the lines: "NLS"="C:\\jvopnav\\win32\\mri\\MessageQueuesMRI.dll" and "JavaPath"="C:\\jvopnav"3. Replace "C:\\\\" with the fully qualified path to the jvopnav directory on your workstation. You must double all back slashes in the path.4. Save your changes.5. Double-click the MsgQueueSampleX.reg registry file. The registry file runs, and the entries from the file are written to the Windows registry on your workstation.
Run the sample Java plug-in.	<ol style="list-style-type: none">1. From System i Navigator, expand the system that you want to scan.2. System i Navigator detects the changes to the registry, and prompts you to scan the system to verify that it is capable of supporting the new plug-in. Click Scan Now.3. System i Navigator scans the system. When the scan finishes, System i Navigator displays a new folder in the hierarchy tree, Java Message Queue Sample 1, 2 or 3. Double click the new folder.4. The first sample plug-in displays the contents of your default message queue in the QUSRSYS library on the system. The second and third samples display a list of message queues. To add a new message, right-click the message queue folder and select New → Message. Enter the message text in the dialog that the plug-in displays. To delete a message, right-click a message and select Delete.5. If you are using the third sample plug-in, you can select a message and drag it to another queue. The plug-in then moves the message to the other queue.

Related information



IBM Client Access Express Toolkit - System i Navigator Plug-ins Web page

Sample Java plug-in directory of files

These tables describe all of the files included with the sample Java plug-ins.

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 have the prefix Mq to differentiate them from like-named classes in other packages.

Java source code files; first sample plug-in	Description
MqActionsManager.java	The ActionsManager implementation that handles all context menus for the plug-in.
MqDeleteMessageBean.java	The UI DataBean implementation for the Confirm Delete dialog.
MqMessage.java	An object representing a system message.
MqMessageQueue.java	A collection of system message objects on a message queue.
MqMessagesListManager.java	The ListManager for lists of messages.
MqNewMessageBean.java	The UI DataBean implementation for the New Message dialog.

Java source code files; second sample plug-in	Description
MqActionsManager.java	The ActionsManager implementation that handles all context menus for the plug-in.
MqDeleteMessageBean.java	The UI DataBean implementation for the Confirm Delete dialog.
MqListManager.java	The master ListManager implementation for the plug-in.
MqMessage.java	An object representing a system message.
MqMessageQueue.java	A collection of system message objects on a particular queue.
MqMessageQueueList.java	A collection of system message queues.
MqMessageQueuesListManager.java	A slave ListManager for lists of message queues.
MqMessagesListManager.java	A slave ListManager for lists of messages.
MqNewMessageBean.java	The UI DataBean implementation for the New Message dialog.

Java source code files; third sample plug-in	Description
MqActionsManager.java	The ActionsManager implementation that handles all context menus for the plug-in.
MqDeleteMessageBean.java	The UI DataBean implementation for the Confirm Delete dialog.
MqDropTargetManager.java	The DropTargetManager implementation that handles drag/drop for the plug-in.
MqListManager.java	The master ListManager implementation for the plug-in.
MqMessage.java	An object representing a system message.
MqMessageQueue.java	A collection of system message objects on a particular queue.
MqMessageQueueList.java	A collection of system message queues.
MqMessageQueuesListManager.java	A slave ListManager for lists of message queues.
MqMessagesListManager.java	A slave ListManager for lists of messages.
MqNewMessageBean.java	The UI DataBean implementation for the New Message dialog.

PDML files	Description
MessageQueueGUI.pdml	Contains all Java UI panel definitions for the plug-in.
MessageQueueGUI.java	The associated Java resource bundle (subclasses java.util.ListResourceBundle).

Online help files	Description
IDD_MSGQ_ADD.html	Online help skeleton for the New Message dialog.
IDD_MSGQ_CONFIRM_DELETE.html	Online help skeleton for the Confirm Delete dialog.

Serialized files	Description
IDD_MSGQ_ADD.pdml.ser	Serialized panel definition for the New Message dialog.
IDD_MSGQ_CONFIRM_DELETE.pdml.ser	Serialized panel definition for the Confirm Delete dialog. Note: If you make changes to MessageQueueGUI.pdml, rename these files. Otherwise your changes will not be reflected in the panels.

Registry entries	Description
MsgQueueSample1.reg MsgQueueSample2.reg MsgQueueSample3.reg	Windows registry entries that tell System i Navigator that this plug-in exists, and identifies its Java interface implementation classes.
MsgQueueSample1install.reg MsgQueueSample2install.reg MsgQueueSample3install.reg	The registry file for distribution with the retail version of your plug-in. This version of the registry file cannot be read directly by the Windows operating system. It contains substitution variables that represent the directory path of the System i Access for Windows installation directory. When the user starts the Install Plug-ins wizard to install your plug-in from the system, the wizard reads this registry file, fills in the correct directory paths, and writes the entries to the registry on the user's workstation. The entries in this file should, therefore, be kept in synchronization with the registry file used in development.

Plug-in programming reference

System i Navigator handles plug-ins in each programming language in a different way.

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

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

C++ reference

C++ plug-ins have a unique flow of control in System i Navigator. You can use a variety of System i Navigator APIs to develop C++ plug-ins. Each plug-in can implement one or more Component Object Model (COM) interfaces.

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

The internal architecture of the System i Navigator product is intended to serve as an integration point for an extensible, broad-based operations interface for the System i platform.

Each functional component of the interface is packaged as an ActiveX server DLL. System i 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 startup, which can consume the majority of Windows resources and impact the performance of the entire system. Multiple systems can register their request to add menu items and dialogs to a given object type in the System i Navigator hierarchy.

Plug-ins work by responding to method calls from System i Navigator that are generated in response to user actions. For example, when a user right-clicks on an object in the System i Navigator hierarchy,

System i Navigator constructs a context menu for the object, and displays the menu on the screen. System i 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 System i 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 can 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.

System i 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 System i Navigator can call to perform a specific function. A plug-in can 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, System i Navigator constructs a context menu for the object and displays it. System i Navigator obtains the menu items by calling each plug-in that has registered that it supplies context menu items for the selected object type. The plug-ins pass their menu items to System i Navigator when it calls their implementation of the **QueryContextMenu** method on the **IContextMenu** interface.

Interface	Method	Description
IContextMenu	QueryContextMenu	Supplies context menu items when a user right-clicks an object.
	GetCommandString	Supplies help text for context menu items and, based on the state of the object, also indicates whether the item should be enabled or not.
	InvokeCommand	Displays the appropriate dialog and performs the requested action. It's called when the user clicks a given menu item.
IPropSheetExt	AddPages	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.

Interface	Method	Description
IDropTarget	DragEnter	Active when the user drags an object over the drop area.
	DragLeave	Active when the user drags an object out of the drop area.
	DragOver	Active while the user is over the drop area.
	Drop	Active when the user drops the object.
IPersistFile	Load	Called to initialize the extension with the fully qualified object name of the selected folder.
IA4SortingHierarchyFolder	IsSortingEnabled	Indicates whether sorting is enabled for a folder.
	SortOnColumn	Sorts the list on the specified list view column.
IA4FilteringHierarchyFolder	GetFilterDescription	Returns a text description of the current include criteria.
IA4PublicObjectHierarchyFolder	GetPublicListObject	Implemented by a plug-in when it desires to make its list objects available for use by other by other plug-ins
IA4ListObject	GetAttributes	Returns a list of supported attribute IDs and the type of data associated with each.
	GetValue	Given an attribute ID, returns the current value of the attribute.
IA4TasksManager	QueryTasks	Returns a list of tasks supported by this object
	TaskSelected	Informs the IA4TasksManager implementation that a particular task has been selected by the user.

IA4 interfaces

In addition to Microsoft's COM interfaces, IBM supplies the IA4HierarchyFolder and IA4PropSheetNotify interfaces.

The IA4PropSheetNotify interface notifies third-party property pages when the main dialog closes. It also defines methods that communicate information to the plug-in. For example, the method can communicate whether the user whose properties are being displayed already exists or is being defined, and whether changes should be saved or discarded.

The IA4HierarchyFolder interface allows a plug-in to add new folders to the System i Navigator hierarchy. The purpose of this interface is to supply the data that is used to populate the contents of a new folder that your plug-in added to the System i 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.

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 System i Navigator hierarchy.

For a description of the Microsoft COM, see the Microsoft Web site.

The System i 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 System i Navigator with list data that will be used when System i Navigator displays the contents of a folder defined by the plug-in. The methods on the interface allow System i Navigator to bind to a particular third-party folder and list its contents. There are methods for returning the number of columns in the details view and their associated headings. There are additional methods 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). System i 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. System i Navigator then 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 codes.

Related information



Microsoft Web site

IA4HierarchyFolder interface specifications listing:

An item identifier data entity 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 under the Desktop on the Windows explorer.

An item identifier consists of a 2-byte count field, 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.

System i Navigator uses item identifiers that adhere to the following structure that must be returned by IA4HierarchyFolder::ItemAt.

```
<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.

IA4HierarchyFolder::Activate:

This specification 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 system 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

```
HRESULT STDMETHODCALLTYPE Activate();
```

Parameters

None.

Return Codes

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

Comments

System i 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 operation of the folder contents.

The function can be called whenever a pointer to the folder interface needs to be reestablished; for example, when a user selects a folder a second time. After another folder is selected, the function should simply return TRUE if the associated processing has already been performed.

For extremely large lists, you might choose to return from the Activate method 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:

This specification returns an instance of IA4HierarchyFolder that corresponds to a particular folder in the System i Navigator hierarchy. This is a required member function.

Syntax

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

Parameters

hwnd The handle of the view window that displays the list, which can be either a tree or list control. A component should use this handle to determine whether a list of objects for this view is already stored in the cache on the client.

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

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

ppvOut

An 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 instance in the cache instead of instantiating and initializing a separate instance. However, if the window handle associated with the object in the cache 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:

This specification is called to display an error message to the end user whenever the `Activate` method returns an error. This is a required member function.

Syntax

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

Parameters

None.

Return Codes

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

Comments

None.

IA4HierarchyFolder::GetAttributesOf:

This specification returns the attributes of a particular folder in the System i 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

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

Parameters

pidl A 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 is 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.

System i Navigator repeatedly calls this function when populating a tree or list view. Long running operations should therefore be avoided.

IA4HierarchyFolder::GetColumnDataItem:

This specification returns a data field for a folder or object to be displayed in a column in the list view of System i Navigator. This is a required member function.

Syntax

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

Parameters

pidl A 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

The address of the buffer that receives the null-terminated data string.

cchMax

The size of the buffer that receives the null-terminated data string.

Return Codes

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

Comments

System i Navigator repeatedly calls this function when populating a list view. Long running operations should therefore be avoided.

IA4HierarchyFolder::GetColumnInfo:

This specification 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

```
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 A4hyfColumnInfoItem 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

System i 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, System i Navigator inserts two columns: Name and Description. The GetColumnDataItem function must be capable of returning data for these two fields, which are identified with integer values of 0 and 1.

Use the Windows IMalloc interface to allocate memory for the returned structures. System i Navigator is responsible for deleting this memory.

IA4HierarchyFolder::GetIconIndexOf:

This specification returns the index into the component resource DLL that can 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** A pointer to an ITEMIDLIST (item identifier list) structure that uniquely identifies the object whose icon index is to be retrieved.
- uFlags** The specification of the type of icon index to retrieve. This parameter might be zero, or it might contain the value GIL_OPENICON, indicating that the icon that should be supplied is an open folder. GIL_OPENICON is defined in the Windows include file SHLOBJ.H.
- piIndex**
A 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

System i Navigator repeatedly calls this function when populating a tree or list view. Long running operations should therefore be avoided.

IA4HierarchyFolder::GetItemCount:

This specification returns the total count of objects contained in a particular folder in the System i Navigator hierarchy. This is a required member function.

Syntax

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

Parameters

- pCount**
A pointer to a long integer that receives 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 the list is being constructed. In this situation, the item count represents only the items that are already stored in the cache on the client.

Comments

Following a successful return from the Activate method, System i 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, System i Navigator repeatedly calls ItemAt to obtain the item identifiers for the objects in the folder.

For extremely large lists, you can choose to return from the Activate function before the entire list has been stored in the cache on the client. If this is the case, you need to return A4HYF_OK_LISTNOTCOMPLETE from the GetItemCount function. From that point on, System i Navigator calls the GetItemCount function every 10 seconds until A4HYF_OK_LISTCOMPLETE or A4HYF_E_LISTDATAERROR is returned.

IA4HierarchyFolder::GetToolBarInfo:

This specification returns a structure that describes the custom toolbar that is associated with the specified folder in the System i Navigator hierarchy. This is a required member function.

Syntax

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

Parameters

pidl A pointer to an ITEMIDLIST (item identifier list) structure that uniquely identifies the object for which toolbar 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 the header file 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 a System i Navigator plug-in.

Use the Windows IMalloc interface to allocate memory for the returned structure. System i Navigator is responsible for deleting this memory.

If this member function is not implemented, the default System i Navigator toolbar is used. This toolbar contains Copy, Paste, Delete, and Properties buttons for the four list views, and Refresh. System i 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 should be enabled for your objects.

IA4HierarchyFolder::GetListObject:

Given a fully qualified object name, this function returns a pointer to a proxy object (created by the plug-in) in the cache. 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 the `cwbUN_GetListObjectFromName` or `cwbUN_GetListObjectFromPidl` API to obtain a proxy object that was instantiated by the `Activate` method. The plug-in uses this proxy object to access data on the system, or to perform actions on the system. Because the `IA4HierarchyFolder` implementation maintains the cache of proxy objects, the calling program should not delete the object.

IA4HierarchyFolder::ItemAt:

This specification 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

An address of the pointer that receives 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

System i 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 System i Navigator item identifiers. Use the Windows IMalloc interface to allocate memory for the item identifier.

IA4HierarchyFolder::ProcessTerminating:

This function is called when the user closes the System i Navigator window. It allows the plug-in 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

None

IA4HierarchyFolder::Refresh:

This specification destroys any folder objects that are stored in the cache and rebuilds the cache using new data obtained from the system. 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

System i Navigator calls this function is called whenever a performing a global refresh of the main System i Navigator window.

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 Component Object Model (COM) interface that IBM defined to allow third parties to add new property pages to any property sheet that System i Navigator defines for a user.

The System i 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 a 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. System i 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. System i 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. These notifications are needed when you add 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 might occur many times before the user clicks **OK** on the main Properties dialog. The IA4PropSheetNotify interface informs the IShellPropSheetExt implementation when changes that are made by the user should be saved.

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

Related concepts

"Property pages for a property sheet handler" on page 84

The Microsoft Foundation Class (MFC) Library classes do not support the creation of property pages for a property sheet handler. However, you can use IBM-provided CExtPropertyPage in place of the MFC class CPropertyPage.

IA4PropSheetNotify::ApplyChanges:

This function is 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

The name of the new user if the user is being created 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

None

IA4PropSheetNotify::GetErrorMessage:

This function is called when errors are returned on the `ApplyChanges` function to retrieve the implementation's error message text.

Syntax

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

Parameters

pszErrMsg

An address of the buffer that receives the null-terminated error message.

cchMax

The size of the buffer that receives 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

None

IA4PropSheetNotify::InformUserState:

This function is called immediately following the creation of the `IShellPropSheetExt` instance. It informs the implementation whether this user already exists on the system or is being created for the first time.

Syntax

```
HRESULT STDMETHODCALLTYPE InformUserState(  
    UINT wUserState  
);
```

Parameters

wUserState

The current state of the user. The system supplies these mutually exclusive values:

- `IUS_NEWUSER`
Creating a user based on attributes that are supplied by the System i Navigator user.
- `IUS_NEWUSERBASEDON`
Creating a user based on the attributes of an existing user.

- IUS_USEREXISTS
The user already exists on the system.

Return Codes

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

Comments

None

System i Navigator APIs

System i Navigator APIs help plug-in developers obtain and manage certain types of global information.

System i Navigator API listing:

The table lists System i Navigator APIs grouped by function.

Function	System i Navigator APIs
System values: This API allows the plug-in developer to obtain the current value of a system value.	"cwbUN_GetSystemValue" on page 34
System handles: These APIs allow the plug-in developer to obtain and release the current value of a system object handle that contains connection properties including the Secure Sockets Layer (SSL) settings to be used for the specified system.	"cwbUN_GetSystemHandle" on page 35 "cwbUN_ReleaseSystemHandle" on page 36
User input validation: These APIs allow the plug-in developer to check whether the current user has authority to a particular System i object. The APIs also allow the developer to determine whether the user has one or more special authorities.	"cwbUN_CheckObjectAuthority" on page 36 "cwbUN_CheckSpecialAuthority" on page 37
User authority checking: This API allows the plug-in developer to check whether certain types of user-supplied strings are valid before transmitting them to the system.	"cwbUN_CheckAS400Name" on page 37
User profile attributes: This API allows the plug-in developer to obtain the value of any of the user profile attributes for the current System i Navigator user.	"cwbUN_GetUserAttribute" on page 38

Function	System i Navigator APIs
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.	“cwbUN_ConvertPidlToString” on page 39 “cwbUN_GetDisplayNameFromItemId” on page 39 “cwbUN_GetDisplayNameFromName” on page 40 “cwbUN_GetDisplayPathFromName” on page 41 “cwbUN_GetIndexFromItemId” on page 41 “cwbUN_GetIndexFromName” on page 42 “cwbUN_GetIndexFromPidl” on page 42 “cwbUN_GetListObject” on page 42 “cwbUN_GetParentFolderNameFromName” on page 43 “cwbUN_GetParentFolderPathFromName” on page 43 “cwbUN_GetParentFolderPidl” on page 44 “cwbUN_GetSystemNameFromName” on page 44 “cwbUN_GetSystemNameFromPidl” on page 45 “cwbUN_GetTypeFromName” on page 46 “cwbUN_GetTypeFromPidl” on page 46
Refresh the System i 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 System i Navigator status bar.	“cwbUN_RefreshAll” on page 47 “cwbUN_RefreshList” on page 47 “cwbUN_RefreshListItems” on page 47 “cwbUN_UpdateStatusBar” on page 48
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 System i Navigator.	“cwbUN_GetODBCConnection” on page 48 “cwbUN_EndODBCConnections” on page 49
Access System i Navigator icons: These APIs allow the plug-in developer to access the icon image lists for objects that appear in the System i Navigator object hierarchy.	“cwbUN_GetIconIndex” on page 49 “cwbUN_GetSharedImageList” on page 50
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 <i>Administrable function</i> is any function whose use can be controlled through the Application Administration subcomponent of System i Navigator.	“cwbUN_GetAdminCacheState” on page 52 “cwbUN_GetAdminCacheStateEx” on page 53 “cwbUN_GetAdminValue” on page 50 “cwbUN_GetAdminValueEx” on page 51
Install: This API allows the plug-in developer to determine if a System i Navigator subcomponent is installed.	“cwbUN_IsSubcomponentInstalled” on page 54

Function	System i Navigator APIs
<p>Directory Services: These APIs provide information about the Lightweight Directory Access Protocol (LDAP) server on a System i platform. These APIs also provide functions to connect to the server. The connection functions enable you to connect to a server using information that System i Access for Windows stores in the cache, such as distinguished names and a password. The connection functions use the LDAP client that is included with System i Access for Windows (LDAP.LIB and LDAP.DLL) and therefore require that your application use that client.</p> <p>Functions that use strings are available in American National Standards Institute (ANSI) and Unicode versions.</p> <p>Functions that return distinguished names and other strings for use with LDAP client APIs are provided in a UTF-8 version for use with LDAP version 3 servers.</p>	<p>“cwbUN_FreeLdapPublishing” on page 58</p> <p>“cwbUN_OpenLdapPublishing” on page 57</p> <p>“cwbUN_GetLdapPublishCount” on page 58</p> <p>“cwbUN_GetLdapPublishParentDn” on page 61</p> <p>“cwbUN_GetLdapPublishPort” on page 60</p> <p>“cwbUN_GetLdapPublishServer” on page 60</p> <p>“cwbUN_GetLdapPublishType” on page 59</p> <p>“cwbUN_GetLdapSvrPort” on page 55</p> <p>“cwbUN_GetLdapSvrSuffixCount” on page 56</p> <p>“cwbUN_GetLdapSvrSuffixName” on page 56</p> <p>“cwbUN_FreeLocalLdapServer” on page 55</p> <p>“cwbUN_OpenLocalLdapServer” on page 54</p>

cwbUN_GetSystemValue:

This API returns a string that contains a 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 system value to be retrieved. Definitions for the system value constants are in the header file CWBA4SVL.H.

char * szSystemValue - output

An address of the buffer that receives the null-terminated system value string.

UINT cchMax - input

The size of the buffer that receives 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_GetSystemHandle:

This API returns a system handle that contains the security, user ID, and password settings that are used for the system. The system handle has the settings that are configured in System i Navigator for the input 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 needs a unique i5/OS job for a system, then NULL or a unique name should be passed for the application name.

If an application needs to share an i5/OS job, then all callers of this function should pass the same application name.

Syntax

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

Parameters

char * szSystemName - input

A 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

A 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

A 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 System i Access for Windows APIs. For example, all System i Access for Windows 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 are released when the System i Navigator application (cwbunnav.exe) ends.

cwbUN_ReleaseSystemHandle:

This API releases a system handle that contains the security settings to be used for the system. The system handle is obtained using the `cwbUN_GetSystemHandle` function. If the caller of this function has the last reference to the handle, the handle resources will be destroyed.

Syntax

```
CWBAPI unsigned int WINAPI cwbUN_ReleaseSystemHandle(  
    cwbCO_SysHandle * systemHandle  
);
```

Parameters

cwbCO_SysHandle * systemHandle - input

A pointer to the handle of the system that was obtained on a `cwbUN_GetSystemHandle` call.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWB_API_ERROR

The system handle could not be released.

Usage When the caller of this function no longer needs the system handle for communications, the handle can be released.

cwbUN_CheckObjectAuthority:

This API returns an indication of whether the System i Navigator user has the authority to a particular object on the system.

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 System i object path for which authority is to be checked.

const char * szObjectType - input

The System i object type of the object for which authority is to be checked; for example, *DTAQ.

const char * szAuthorityType - input

The System i object authority to be checked.

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

const char * szSystemName - input

The name of the 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 can be specified. *AUTLMGT is valid only if szObjectType is *AUTL.

cwbUN_CheckSpecialAuthority:

This API returns an indication of whether the System i Navigator user has a particular special authority on the system.

Syntax

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

Parameters**const char * szSpecialAuthority - input**

The System i special authority to be checked.

const char * szSystemName - input

The name of the 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 None

cwbUN_CheckAS400Name:

This API returns an indication of whether a specified string is a valid name parameter on the system.

Syntax

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

Parameters**const char * szAS400Name - input**

The system name whose validity is to be checked.

const char * szSystemName - input

The name of the 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

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 None

cwbUN_GetUserAttribute:

This API returns a string that contains the value of a user profile attribute for the current System i Navigator user.

Syntax

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

Parameters**USHORT usAttributeId - input**

A numeric value that 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 system from which to retrieve the user attribute.

char * szValue - output

An address of the buffer that receives the null-terminated attribute value string.

UINT cchMax - input

The size of the buffer that receives 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 instead of 'None.'

cwbUN_ConvertPidlToString:

This API converts an item identifier list in System i Navigator to a fully qualified object name.

Syntax

```
CWBAPI unsigned int WINAPI cwbUN_ConvertPidlToString(  
    LPCITEMIDLIST pidl,  
    char * szObjectName,  
    UINT cchMax  
);
```

Parameters

LPCITEMIDLIST pidl - input

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

char * szObjectName - output

An address of the buffer that receives the null-terminated object name.

UINT cchMax - input

The size of the buffer that receives 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 None

cwbUN_GetDisplayNameFromItemId:

This API extracts the item name field from a Unity item identifier.

Syntax

```
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 is extracted.

char * szItemName - output

An address of the buffer that receives the null-terminated item name.

UINT cchMax - input

The size of the buffer that receives the null-terminated item name.

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 None

cwbUN_GetDisplayNameFromName:

This API 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 is extracted.

char * szItemName - output

An address of the buffer that receives the null-terminated item name.

UINT cchMax - input

The size of the buffer that receives 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 None

cwbUN_GetDisplayPathFromName:

This API converts a fully qualified Unity object name to a fully qualified path name 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 path name is derived.

char * szPathName - output

An address of the buffer that receives the null-terminated path name.

UINT cchMax - input

The size of the buffer that receives the null-terminated path 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 None

cwbUN_GetIndexFromItemId:

This API extracts the item index field from a Unity item identifier.

Syntax

```
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 is extracted.

ULONG* piIndex - output

An address of an unsigned long integer that receives 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 None

cwbUN_GetIndexFromName:

This API extracts the item index field from a fully qualified Unity object name.

Syntax

```
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 is extracted.

ULONG* piIndex - output

An address of an unsigned long integer that receives 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 None

cwbUN_GetIndexFromPidl:

This API extracts the item index field from a fully qualified Unity item identifier list.

Syntax

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

Parameters

LPCITEMIDLIST pidl - input

A pointer to an ITEMIDLIST (item identifier list) structure from which the item index is extracted.

ULONG* piIndex - output

An address of an unsigned long integer that receives 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 None

cwbUN_GetListObject:

This API gets 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 is found and returned.

LPVOID pListObject - output

An address of a pointer to the request Unity object.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

Usage None

cwbUN_GetParentFolderNameFromName:

This API 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 is extracted.

char * szParentFolderPath - output

An address of the buffer that receives the null-terminated parent folder name.

UINT cchMax - input

The size of the buffer that receives 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.

CWB_BUFFER_OVERFLOW

The buffer is too small to contain the returned string.

Usage None

cwbUN_GetParentFolderPathFromName:

Given a fully qualified Unity object name, this API returns the fully qualified object name of the object's parent folder.

Syntax

```
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 is extracted.

char * szParentFolderPath - output

An address of the buffer that receives the null-terminated parent folder object name.

UINT cchMax - input

The size of the buffer that receives 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 None

cwbUN_GetParentFolderPidl:

Given a fully qualified Unity item identifier list, this API returns the fully qualified item identifier list of the object's parent folder.

Syntax

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

Parameters

LPCITEMIDLIST pidl - input

A pointer to an ITEMIDLIST (item identifier list) structure from which the parent folder item identifier list is extracted.

LPITEMIDLIST* ppidl - output

An address of an item identifier list pointer that receives 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 None

cwbUN_GetSystemNameFromName:

This API extracts the 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 is extracted.

char * szSystemName - output

An address of the buffer that receives the null-terminated system name.

UINT cchMax - input

The size of the buffer that receives 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 None

cwbUN_GetSystemNameFromPidl:

This API extracts the 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

A pointer to an ITEMIDLIST (item identifier list) structure from which the system name is extracted.

char * szSystemName - output

An address of the buffer that receives the null-terminated system name.

UINT cchMax - input

The size of the buffer that receives 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 None

cwbUN_GetTypeFromName:

This API 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 is extracted.

char * szType - output

An address of the buffer that receives the null-terminated item type.

UINT cchMax - input

The size of the buffer that receives 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 None

cwbUN_GetTypeFromPidl:

This API 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

A pointer to an ITEMIDLIST (item identifier list) structure from which the item index is extracted.

char * szType - output

An address of the buffer that receives the null-terminated item type.

UINT cchMax - input

The size of the buffer that receives 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 None

cwbUN_RefreshAll:

This API refreshes the contents of the tree window and the list window for System i Navigator.

Syntax

```

CWBAPI unsigned int WINAPI cwbUN_RefreshAll(
    const char * pszStatusText
);

```

Parameters**const char * pszStatusText - input**

A null-terminated string to be placed in the status bar window on completion. This parameter can be NULL.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWBUN_WINDOW_NOTAVAIL

Could not find the view windows.

Usage Use this function to refresh the entire contents of System i Navigator after the system performs an action that is requested by the user.

cwbUN_RefreshList:

This API refreshes the contents of the list view window for System i Navigator.

Syntax

```

CWBAPI unsigned int WINAPI cwbUN_RefreshList(
    const char * pszStatusText
);

```

Parameters**const char * pszStatusText - input**

A null-terminated string to be placed in the status bar window on completion. This parameter can be NULL.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWBUN_WINDOW_NOTAVAIL

Could not find list view window.

Usage Use this function to refresh the contents of the list window after performing an action that is requested by the user.

cwbUN_RefreshListItems:

This API refreshes the currently selected item (or items) in the list view window of System i Navigator.

Syntax

```
CWBAPI unsigned int WINAPI cwBUN_RefreshListItems(  
    const char * pszStatusText  
);
```

Parameters

const char * pszStatusText - input

A null-terminated string to be placed in the status bar window on completion. This parameter can be NULL.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWBUN_WINDOW_NOTAVAIL

Could not find list view window.

Usage Use this function to refresh the selected items in the list window after performing an action that was requested by the user.

cwBUN_UpdateStatusBar:

This API inserts a text string into the status bar of System i Navigator window.

Syntax

```
CWBAPI unsigned int WINAPI cwBUN_UpdateStatusBar(  
    const char * pszStatusText  
);
```

Parameters

const char * pszStatusText - input

A null-terminated string to be placed in the status bar window on completion.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWBUN_WINDOW_NOTAVAIL

Could not find status bar window.

Usage Use this function to inform the user that an action that was requested by clicking the OK button on a dialog has completed successfully.

cwBUN_GetODBCConnection:

This API returns the handle to an Open Database Connectivity (ODBC) connection on the specified system. If no connection exists to the specified system, 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 system on which to retrieve an ODBC connection.

HDBC *phDBC - output

The address to return the ODBC connection handle.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

Usage None

cwbUN_EndODBCConnections:

This API ends all Open Database Connectivity (ODBC) connections previously opened by the `cwbUN_GetODBCConnection` API.

Syntax

```
CWBAPI unsigned int WINAPI cwbUN_EndODBCConnections(
);
```

Parameters

None

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_HANDLE

Handle was not created.

Usage

It is important to remember that the **EndODBCConnections** function only closes connections that were opened using the **GetODBCConnection** function. The **EndODBCConnections** 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 **EndODBCConnections** if any code in your extension uses **GetODBCConnection**.

See also `cwbUN_GetODBCConnection`.

cwbUN_GetIconIndex:

This API gets 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**

A pointer to the ITEMIDLIST (item identifier list) structure that is used to identify the icon to be referenced.

UINT uFlags - input

The specification of the type of icon index to retrieve (defined above).

int * piIndex - output

An address of the integer that receives 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 None

cwbUN_GetSharedImageList:

This API retrieves the icon image list associated with System i Navigator.

Syntax

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

Parameters

UINT uFlags - input

The specification of the type of image list to retrieve (defined above).

HIMAGELIST* phImageList -

An address of the variable that receives 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 None

cwbUN_GetAdminValue:

This API returns an indication of whether the current System i Navigator user on the specified system 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 System i Navigator.

For example, an administrator can use Application Administration to control whether a user can access several functions in System i Navigator. One of these functions is job management. The `cwbUN_GetAdminValue` API can be used to programmatically determine whether the current System i 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 System i Navigator.

Syntax

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

Parameters

const char * szSystemName

The name of the 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 whether the current System i Navigator user for the specified system is allowed to use the specified function. If no user is currently signed on to the specified system, the API signs the user on, possibly displaying a user ID and password prompt.

This API can only be used to check administrable functions that are in System i Navigator or in the Client Applications function category.

cwbUN_GetAdminValueEx:

This API returns an indication of whether the current user on the specified system 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 System i Navigator.

Note: System i Navigator plug-ins should use the cwbUN_GetAdminValue API instead of cwbUN_GetAdminValueEx.

An administrator can use Application Administration to control whether a user can access several functions in System i Navigator. One of these functions is job management. The cwbUN_GetAdminValueEx API can be used to programmatically determine whether 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 System i 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

```
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 before this API is called. The `cwbUN_GetAdminValueEx` API's behavior is based on whether the system object has obtained a sign-on to the system:

Not Signed On->

`cwbUN_GetAdminValueEx` signs on to the system. The latest Application Administration settings for the user are downloaded from the system if they are not already stored in the cache on the client workstation.

Signed On->

If the system object was signed on to a system that specifies that the System i user ID and password should be validated (Validate Mode), then the `cwbUN_GetAdminValueEx` API uses a snapshot of Application Administration settings that were accurate when the sign-on was completed. If the sign-on was done without validating the user ID and password, then it is possible that `cwbUN_GetAdminValueEx` is using a copy of the Application Administration settings that might be as much as 24 hours old.

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 whether the current system 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 system, the API signs the user on, possibly displaying a user ID and password prompt.

This API can only be used to check Administrable functions that are in System i Navigator or in the Client Applications function category.

cwbUN_GetAdminCacheState:

This API indicates whether the next invocation of the `cwbUN_GetAdminValue` API is long running. The `cwbUN_GetAdminValue` API stores data in the cache on the workstation. If the cache is not current, `cwbUN_GetAdminValue` can present a sign-on prompt or perform other processing to update its cache.

Syntax

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

Parameters

const char * szSystemName

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

cwbUN_State& adminState

A parameter that indicates whether the next invocation of the cwbUN_GetAdminValue API is long running, or whether it uses its internal cache to return without accessing the host system.

One of these values is returned:

cwbUN_logon

There is no current user for the specified system. The cwbUN_GetAdminValue API can present a sign-on prompt.

cwbUN_refresh

cwbUN_GetAdminValue accesses the system 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 Users of cwbUN_GetAdminValue can use this API to determine whether the next invocation of cwbUN_GetAdminValue is long running.

cwbUN_GetAdminCacheStateEx:

This API indicates whether the next invocation of the cwbUN_GetAdminValueEx API is long running. The cwbUN_GetAdminValueEx API stores data in the cache on the workstation. If the cache is not current, the cwbUN_GetAdminValueEx API can present a sign-on prompt or perform other processing to update its cache.

Syntax

```
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

A parameter that indicates whether the next invocation of the cwbUN_GetAdminValue API is long running, or whether it uses its internal cache to return without accessing the host system.

One of these values is returned:

cwbUN_logon

There is no current user for the specified system. The cwbUN_GetAdminValue API can present a sign-on prompt.

cwbUN_refresh

cwbUN_GetAdminValue accesses the system 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 Users of `cwbUN_GetAdminValueEx` can use this API to determine whether the next invocation of `cwbUN_GetAdminValueEx` is long running.

cwbUN_IsSubcomponentInstalled:

This API determines whether a System i Navigator subcomponent is installed on the workstation.

Syntax

```
CWBAPI BOOL WINAPI cwbUN_IsSubcomponentInstalled(
    UNIT uOption);
```

Parameters

UNIT uOption

This parameter specifies the System i Navigator subcomponent to check. See the API's prolog in `cwbun.h` for a list of supported values.

Return Codes

Returns a boolean value.

TRUE If the subcomponent is installed.

FALSE

If the subcomponent is not installed.

Usage None.

cwbUN_OpenLocalLdapServer:

This API creates a handle that can be used to access configuration information about the Lightweight Directory Access Protocol (LDAP) server on the system.

Syntax

```
int cwbUN_OpenLocalLdapServerW
( LPCWSTR      system,
  cwbUN_ldapSvrHandle *pHandle
);

int cwbUN_OpenLocalLdapServerA
( LPCSTR      system,
  cwbUN_ldapSvrHandle *pHandle
);
```

Parameters

LPCSTR system - input

A pointer to the system name.

cwbUN_ldapSvrHandle *pHandle - output

On return, contains a handle that can be used with the following APIs:

- `cwbUN_FreeLocalLdapServer`
- `cwbUN_GetLdapSvrPort`
- `cwbUN_GetLdapSvrSuffixCount`
- `cwbUN_GetLdapSuffixName`

Note: This handle should be released with a call to `cwbUN_FreeLocalLdapServer`.

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.

Usage None

cwbUN_FreeLocalLdapServer:

This API frees resources associated with the input handle.

Syntax

```
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_GetLdapSvrPort:

This API returns the port number that is used by the Lightweight Directory Access Protocol (LDAP) server.

Syntax

```
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 None

cwbUN_GetLdapSvrSuffixCount:

This API 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

```
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 None

cwbUN_GetLdapSvrSuffixName:

This API 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

A zero-based index of the suffix. This value must be less than the count returned by cwbUN_GetLdapSvrSuffixCount().

LPSTR suffix - output

A pointer to the buffer that contains the distinguished name of the suffix.

int * length - input/output

A 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 is 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 None

cwbUN_OpenLdapPublishing:

This API creates a handle that can be used to access configuration information about information that is published by the system to Lightweight Directory Access Protocol (LDAP) directories.

Syntax

```

int cwbUN_OpenLdapPublishingW
( LPCWSTR      system,
  cwbUN_ldapPubHandle *pHandle
);

int cwbUN_OpenLdapPublishingA
( LPCSTR      system,
  cwbUN_ldapPubHandle *pHandle
);

```

Parameters

LPCSTR system - input

A pointer to the system name.

cwbUN_ldapSvrHandle *pHandle - output

On return, contains a handle that can be used with APIs.

Return Codes

The following list shows common return values:

CWB_OK

Successful completion.

CWB_INVALID_API_PARAMETER

Invalid parameter specified.

CWB_INVALID_API_HANDLE

Invalid handle.

CWB_INVALID_POINTER

A NULL pointer was specified.

CWBUN_LDAP_NOT_AVAIL

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

Usage None

cwbUN_FreeLdapPublishing:

This API frees resources associated with the input handle.

Syntax

```
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_GetLdapPublishCount:

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

Syntax

```
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 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 None

cwbUN_GetLdapPublishType:

This API returns the publish record information type.

Syntax

```
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

A 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

System i platforms.

CWBUN_LDAP_PUBLISH_NETWORK_INVENTORY

NetFinity.

CWBUN_LDAP_PUBLISH_PRINTERS

Printers connected to the System i platform.

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 None

cwbUN_GetLdapPublishServer:

This API returns the name of the server to which this information is published.

Syntax

```
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

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

LPWSTR server - output

A pointer to the buffer that contains the name of the server.

int * length - input/output

A 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 is 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 None

cwbUN_GetLdapPublishPort:

This API 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

A 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. This parameter allows these values:

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 None

cwbUN_GetLdapPublishParentDn:

This API 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

```
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

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

LPSTR parentDn - output

A pointer to the buffer that contains the name of the parentDn.

int * length - input/output

A 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 is 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 None

Return codes unique to System i Navigator APIs

System i Navigator has a specific set of return codes. Each code has its own associated meaning.

```

6000  CWBUN_BAD_PARAMETER
      An input parameter was not valid.
6001  CWBUN_FORMAT_NOT_VALID
      The input object name was not valid.
6002  CWBUN_WINDOW_NOTAVAIL
      View window not found.
6003  CWBUN_INTERNAL_ERROR
      Processing error occurred.
6004  CWBUN_USER_NOT_AUTHORIZED
      User does not have specified authority.
6005  CWBUN_OBJECT_NOT_FOUND
      Object not found on the iSeries.
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 iSeries name type.
 6014 CWBUN_INVALID_AUTH_TYPE
 Invalid authority type.
 6016 CWBUN_HOST_COMM_ERROR
 iSeries communications error.
 6017 CWBUN_INVALID_NAME_PARM
 Invalid name parameter.
 6018 CWBUN_NULL_DISPLAY_STRING
 Null display string returned.
 6019 CWBUN_GENERAL_FAILURE
 General iSeries 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 is empty - no chars at all.
 6054 CWBUN_NAME_INVALIDCHAR
 Invalid character.
 6055 CWBUN_NAME_STRINGTOOLONG
 String too long.
 6056 CWBUN_NAME_MISSINGENDQUOTE
 End quote missing.
 6057 CWBUN_NAME_INVALIDQUOTECHAR
 Char invalid for quote string.
 6058 CWBUN_NAME_ONLYBLANKS
 A string of only blanks found.
 6059 CWBUN_NAME_STRINGTOOSHORT
 String is too short.
 6060 CWBUN_NAME_TOOLONGFORIBM
 String OK, too long for IBM^(R) cmd.
 6011 CWBUN_NAME_INVALIDFIRSTCHAR
 The first char is invalid.
 6020 CWBUN_NAME_CHECK_LAST
 Reserved range.

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 iSeries^(TM) 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.

```

Visual Basic reference

Visual Basic plug-ins have a unique flow of control in System i Navigator. In addition, Visual Basic plug-ins must be implemented at least on one System i Navigator interface class.

System i Navigator structure and flow of control for Visual Basic plug-ins

For Visual Basic plug-ins, System i Navigator provides a built-in ActiveX server that manages the communication between System i Navigator and the plug-in.

Visual Basic programmers who are developing System i Navigator plug-ins can use the facilities that are provided by Microsoft's Visual Basic 5.0 to create their plug-in classes and then package them in an ActiveX server DLL.

Plug-ins work by responding to method calls from System i Navigator that are generated in response to user actions. For example, when a user right-clicks an object in the System i Navigator hierarchy, System i Navigator constructs a context menu for the object and displays the menu on the screen. System i 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 System i Navigator can call to perform a specific function. For Visual Basic plug-ins, three interfaces are defined:

- ListManager
- ActionsManager
- DropTargetManager

System i Navigator data for Visual Basic plug-ins

When System i Navigator calls a function implemented by a plug-in, the request typically involves an object or objects that the user selected in the main System i 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 that provides information about the selected objects is defined. Plug-ins that add folders to the object hierarchy must return items in the folder to System i Navigator in the form of item identifiers. For Visual Basic plug-ins, an `ItemIdentifier` class is defined and is used by the plug-in to return the requested information.

System i Navigator services for Visual Basic plug-ins

System i Navigator plug-ins sometimes need to affect the behavior of the main System i Navigator window. For example, following the completion of a user operation, it might be necessary to refresh the System i Navigator list view, or to insert text into the System i Navigator's status area. A utility class called `UIServices`, which provides the required services, is supplied in the Visual Basic environment. A Visual Basic plug-in can also use the C++ APIs in the `cwbun.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 System i Navigator Visual Basic Plug-in Support DLL (`cwbunvbi.dll` and `cwbunvbi.hlp`).

Related concepts

"System i Navigator `ListManager` interface class"

The **ListManager interface class** is used for data serving in System i Navigator. For example, when a list view needs to be created and filled with objects, System i Navigator will call methods in the `ListManager` class to do this.

"System i Navigator `ActionsManager` interface class" on page 66

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 System i Navigator, the `queryActions` method in the `ActionsManager` interface class will be called to return the context menu item strings.

"System i Navigator `DropTargetManager` interface class" on page 66

The **DropTargetManager interface class** is used to handle drag-and-drop operations in System i Navigator.

System i Navigator Visual Basic interfaces

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

The Programmer's Toolkit contains a link to the Visual Basic interface definition help file.

There are three System i Navigator interface classes:

- System i Navigator `ActionsManager` interface class
- System i Navigator `DropTargetManager` interface class
- System i Navigator `ListManager` interface class

Note: Your application does not have to implement all three interface classes.

System i Navigator `ListManager` interface class:

The **ListManager interface class** is used for data serving in System i Navigator. For example, when a list view needs to be created and filled with objects, System i 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 System i Navigator component lists.

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

Related concepts

“System i Navigator structure and flow of control for Visual Basic plug-ins” on page 64
For Visual Basic plug-ins, System i Navigator provides a built-in ActiveX server that manages the communication between System i Navigator and the plug-in.

System i 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 System i 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 System i Navigator Visual Basic Plug-in Support DLL (cwbunvbi.dll and cwbunvbi.hlp files).

Related concepts

“System i Navigator structure and flow of control for Visual Basic plug-ins” on page 64
For Visual Basic plug-ins, System i Navigator provides a built-in ActiveX server that manages the communication between System i Navigator and the plug-in.

System i Navigator DropTargetManager interface class:

The **DropTargetManager interface class** is used to handle drag-and-drop operations in System i 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 System i Navigator Visual Basic Plug-in Support DLL (cwbunvbi.dll and cwbunvbi.hlp).

Related concepts

“System i Navigator structure and flow of control for Visual Basic plug-ins” on page 64
For Visual Basic plug-ins, System i Navigator provides a built-in ActiveX server that manages the communication between System i Navigator and the plug-in.

Java reference

Java plug-ins have a unique flow of control in System i Navigator.

System i Navigator structure and flow of control for Java plug-ins

For Java plug-ins, System i Navigator provides a built-in ActiveX server that manages the communication between System i 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 System i Navigator plug-ins do not need to be concerned with the details of ActiveX server implementation.

When a user is interacting with System i 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 System i Navigator that are generated in response to user actions. For example, when a user right-clicks an object in the System i Navigator hierarchy, System i

Navigator constructs a context menu for the object and displays the menu on the screen. System i 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 System i Navigator can call to perform a specific function. For Java plug-ins, the following three **Java interfaces** are defined:

- ActionsManager
- DropTargetManager
- ListManager

Product architecture for System i Navigator plug-ins

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

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

System i Navigator data for Java plug-ins

When System i Navigator calls a function implemented by a plug-in, the request typically involves an object or objects that the user selected in the main System i 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. It provides information about the selected objects. Plug-ins that add folders to the object hierarchy must return items in the folder to System i Navigator in the form of item identifiers. For Java plug-ins, an `ItemIdentifier` class is defined. It is used by the plug-in to return the requested information.

System i Navigator plug-ins sometimes need to affect the behavior of the main System i Navigator window. For example, following the completion of a user operation, it might be necessary to refresh the System i Navigator list view, or to insert text into the System i Navigator's status area. Utility classes that provide the required services are supplied in the package `com.ibm.as400.opnav`.

Customizing the plug-in registry files

The sample plug-ins include two registry files: a windows-readable copy for use during development, and a copy for distribution on the system. You need to modify these registry files to develop your plug-in. This topic provides an overview of the registry files, and detailed descriptions of the required sections of each registry file.

System i Navigator uses the registry files to learn about the plug-in's existence, requirements, and functions. To provide that information, every plug-in must specify at least the following information:

- A primary registry key that provides global information about the plug-in.
This section includes the Programmatic Identifier (ProgID) that specifies the vendor and component name for your plug-in and names the folder in which your plug-in resides on the system. The ProgID must follow the form `<vendor>.<component>`; for example, `IBM.Sample`.
- Registry keys that identify the object types in the System i Navigator hierarchy for which a plug-in intends to supply additional function.

- A separate registry key for the root of each subtree of objects that a plug-in adds to the object hierarchy.

This key contains information about the root folder of the subtree.

Customizing the C++ registry values

The sample plug-in includes two registry files: SAMDBG.REG, a registry file for use during development, and SAMPRLS.REG, a registry file for distribution on the system. Both files can be read by the Windows operating system. You can customize the sample registry files for your own plug-ins.

A plug-in registry file consists of several sections. When you develop your own plug-ins, you need to customize each section as described in this information.

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-in\IBM.Sample]
"Type"="PLUGIN"
"NLS"="sampmri.dll"
"NameID"=dword:00000080
"DescriptionID"=dword:00000081
"MinimumIMPIRelease"="NONE"
"MinimumRISCRRelease"="030701"
"ProductID"="NONE"
"ServerEntryPoint"="sampect.dll"
```

Primary registry key field	Field description
Type	If the plug-in adds new folders to the System i 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 the 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 System i 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 System i Navigator user interface.

Primary registry key field	Field description
MinimumIMPIRelease	<p>A 6-character string that identifies the minimum release of i5/OS that runs on the IMPI hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the i5/OS 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."</p> <p>If the plug-in does not support any i5/OS 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."</p>
MinimumRISCRelease	<p>A 6-character string that identifies the minimum release of i5/OS that runs on RISC hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the i5/OS 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".</p> <p>If the plug-in does not support any i5/OS 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."</p>
ProductID	<p>A 7-character string that specifies the product ID of a prerequisite System i 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 system, the value of this field should be NONE.</p> <p>Multiple comma-separated product IDs can be specified if multiple IDs exist for the same product.</p>
ServerEntryPoint	<p>The name of the code DLL that implements the server entry point. This entry point is called by System i Navigator when it needs to determine whether the plug-in is supported on a particular system. 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.</p>
JavaPath	<p>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 System i Access for Windows installation path, each preceded by the System i Access for Windows substitution variable that represents the installation path.</p>

Primary registry key field	Field description
JavaMRI	The base names of the JAR files that contain locale-dependent resources for the plug-in. System i 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, System i Navigator will expect the MRI for the base locale (usually US English) to reside in the code JAR files.

Data server implementation:

This section registers an IA4HierarchyFolder implementation for each new folder added to the System i Navigator hierarchy.

```
-----
; This section will register an IA4HierarchyFolder implementation
; for each new
; folder added to the System i Navigator hierarchy.

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

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

If your plug-in adds more than one new folder to the hierarchy, you must duplicate this section of the registry file for each additional folder. Make sure to generate a separate Globally Unique Identifier (GUID) for each folder. If your plug-in does not add any folders, you can remove this section.

If you duplicate SAMPDATA.CPP as follows, all of your new folders initially contain library objects:

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 "Global changes for C++ plug-in registry files" on page 73.
3. Replace both occurrences of the class identifier (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, and then change the CLSID in the IMPLEMENT_OLECREATE macro call to match the hexadecimal 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.

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.

7. 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.
8. To add the new implementation files to the project workspace, open the **Insert** menu and select **Files Into Project**.

Shell plug-in implementation class:

This section registers the shell plug-in implementation class. Every c++ plug-in must use this section.

```
;-----
; This section will register the shell plug-in implementation class.
; A shell plug-in adds context menu items and/or property pages
; for new or existing objects in the hierarchy.
```

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

```
[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(R))
```

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

To customize this section for you own plug-ins, follow these steps:

1. Change the DLL name to match the name of the DLL generated by your new project workspace.
2. Generate and copy a new Globally Unique Identifier (GUID). See "Global changes for C++ plug-in registry files" on page 73.
3. Replace all occurrences of the class identifier (CLSID) in the entries 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, and then change the CLSID in the IMPLEMENT_OLECREATE macro call to match the hexadecimal values in your new GUID.

Shell plug-in implementation for objects:

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

```
;-----
; 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 the Auto Refresh property sheet handler for the new folder and its objects
; (this will allow your folder to take advantage of the System i Navigator
; Auto Refresh function)
```

```
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-inS\IBM.Sample\shellex\Sample\*
\PropertySheetHandlers\{5E44E520-2F69-11d1-9318-0004AC946C18}]
```

```
;-----
; Register drag and drop context menu handlers
```

```
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-inS\IBM.Sample\shellex\Sample\*
\DragDropHandlers\{3D7907A1-9080-11d0-82BD-08005AA74F5C}]
```

```
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-inS\IBM.Sample\shellex\File Systems\*
\DragDropHandlers\{3D7907A1-9080-11d0-82BD-08005AA74F5C}]
```

```

;-----
; Register Drop Handler to accept drops of objects

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-in\IBM.Sample\shellex\Sample*\DropHandler]
@="{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

```

To customize this section for your own plug-ins, follow these steps:

1. Replace the class identifier (CLSID) in this section with the new Globally Unique Identifiers (GUIDs).
2. If your plug-in does 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 is not a drop handler for objects, remove the drop-context menu handler and 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.

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 System i 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 System i Navigator component. For plug-ins that add new folders, this identifier should always match the registry key name for a root folder specified in the previous section. For plug-ins that add behaviors to existing object types, this subkey should generally be the object type of the first-level folder under a System i 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 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 need to 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 System i Navigator when serving data for a folder that is defined by your plug-in.

Global changes for C++ plug-in registry files:

When developing your own plug-ins, you need to make some global changes to the sample plug-in registry files. You must specify a unique programmatic identifier (ProgID) and Globally Unique Identifiers (GUIDs) for use throughout the plug-in registry file.

Defining a unique 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 provided. In the sample plug-in, the string IBM.Sample identifies IBM as the vendor, and Sample as the description of the function provided by the plug-in. This is used throughout the registry file. It names the directory where your plug-in resides on both the system and the workstation. Replace every occurrence of IBM.Sample in the registry file with your ProgID.

Generating new GUIDs and replacing the CLSID values in the registry file

For your System i Navigator C++ plug-in to work properly, you must replace specific class identifiers (CLSIDs) in your new registry file with GUIDs that you generate.

The Component Object Model from Microsoft uses 16-byte hexadecimal integers to uniquely identify ActiveX implementation classes and interfaces. These integers are known as GUIDs. GUIDs that identify implementation classes are called CLSIDs. System i 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, a System i Navigator plug-in must register a CLSID for each implementation class that it provides.

To generate your GUIDs, follow these steps:

1. From the Windows task bar, 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**.

Customizing the Visual Basic plug-in registry values

The sample plug-in includes two registry files: VBSMPDBG.REG, a registry file for use during development, and VBSMPRLS.REG, a registry file for distribution on the system. Both files can be read by the Windows operating system. You can customize the sample registry files for your own plug-ins.

A plug-in registry file consists of several sections. When you develop your own plug-ins, you need to customize each section as described in this information.

Primary registry key:

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

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

```
[HKEY_CLASSES_ROOT\IBM.AS400.Network  
\3RD PARTY_EXTENSIONS\IBM.VBSample]
```

```
"Type"="Plugin"
```

```
"NLS"="vbsmpmri.dll"
```

```
"NameID"=dword:00000080
```

```
"DescriptionID"=dword:00000081
```

```
"MinimumIMPIRelease"="NONE"
```

```
"MinimumRISCRRelease"="040200"
```

```
"ProductID"="NONE"
```

```
"ServerEntryPoint"="vbsample.dll"
```

Primary registry key field	Field description
Type	If the plug-in adds new folders to the System i 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 the 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 System i 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 System i Navigator user interface.
MinimumIMPIRelease	<p>A 6-character string that identifies the minimum release of i5/OS that runs on the IMPI hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the i5/OS 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."</p> <p>If the plug-in does not support any i5/OS 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."</p>

Primary registry key field	Field description
MinimumRISCRelease	<p>A 6-character string that identifies the minimum release of i5/OS that runs on RISC hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the i5/OS 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".</p> <p>If the plug-in does not support any i5/OS 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."</p>
ProductID	<p>A 7-character string that specifies the product ID of a prerequisite System i 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 system, the value of this field should be NONE.</p> <p>Multiple comma-separated product IDs can be specified if multiple IDs exist for the same product.</p>
ServerEntryPoint	<p>The name of the code DLL that implements the server entry point. This entry point is called by System i Navigator when it needs to determine whether the plug-in is supported on a particular system. 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.</p>
JavaPath	<p>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 System i Access for Windows installation path, each preceded by the System i Access for Windows substitution variable that represents the installation path.</p>
JavaMRI	<p>The base names of the JAR files that contain locale-dependent resources for the plug-in. System i 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, System i Navigator will expect the MRI for the base locale (usually US English) to reside in the code JAR files.</p>

To customize the primary registry key for you own plug-ins, follow these steps:

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.

Related concepts

“Global changes for Visual Basic plug-in registry files” on page 79

When developing your own plug-ins, you need to define a unique programmatic identifier (ProgID) for your plug-in. You must specify a unique ProgID for use throughout the file.

Visual Basic plug-in implementation class:

This section registers a Visual Basic Plug-in ListManager class implementation for each new folder added to the System i Navigator hierarchy.

If your plug-in does not add any new folders to the System i Navigator hierarchy, delete this section.

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 a system named AS4 in the System i Navigator hierarchy. If you want your folder to appear at some other point in the hierarchy, you must change the Parent key value.

```
[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
```

To customize this section for your own plug-ins, follow these steps:

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.

Parent field values:

A 3-character ID is used to identify the parent of the folder to be added. System i Navigator provides a set of IDs for the parent key value.

You can specify one of the following IDs:

AS4	System folder
-----	---------------

BKF	Backup folder
BOF	Basic Operations folder
CFG	Configuration and Service folder
DBF	Database folder
FSF	File Systems folder
JMF	Job Management folder
MCN	Management Central folder
MCS	Management Central Configuration and Service folder
MDF	Management Central Definitions folder
MST	Management Central Scheduled Tasks
MSM	Management Central Monitors
MTA	Management Central Task Activity
MXS	Management Central Extreme Support
NSR	Network Servers folder
NWF	Network folder
SCF	Security folder
UGF	Users and Groups folder

Related concepts

“Example: New folder registry key”

You must define a separate registry key for the root of each subtree of objects that a plug-in adds to the object hierarchy. This key contains information specific to the root folder of the subtree. This topic describes each new folder registry key field and possible values.

Example: New folder registry key:

You must define a separate registry key for the root of each subtree of objects that a plug-in adds to the object hierarchy. This key contains information specific to the root folder of the subtree. This topic describes each new folder registry key field and possible values.

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

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

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-in\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"
```

Parent	A three-character ID that identifies the parent of the folder to be added.
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.
CLSID	<p>The CLSID of the IA4HierarchyFolder implementation that should be called by System i Navigator to obtain the contents of the folder.</p> <p>For Java plug-ins, the CLSID always should be: 1827A856-9C20-11d1-96C3-00062912C9B2.</p> <p>For Visual Basic plug-ins, the CLSID should always be: 040606B1-1C19-11d2-AA12-08005AD17735).</p>

JavaClass	The fully qualified Java class name of the ListManager implementation that should be called by System i Navigator to obtain the contents of the folder. This field should be omitted if the plug-in is not a Java plug-in.
VBClass	The Program Identifier (ProgID) of the ListManager implementation class that should be called by System i Navigator to obtain the contents of the folder.
VBInterface	The GUID of the ListManager implementation class' interface.
NameID	A double word that contains the resource ID of the string that should appear as the name of the folder in the System i 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 System i 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 System i 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 System i Navigator hierarchy whenever it is selected by the user.
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.

Related concepts

"Parent field values" on page 76

A 3-character ID is used to identify the parent of the folder to be added. System i Navigator provides a set of IDs for the parent key value.

Visual Basic plug-in implementation objects:

The final section of the registry specifies which objects in the System i 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.

```

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

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\
IBM.VBSample\shellex\SampleVBFolder\*\
ContextMenuHandlers\{040606B2-1C19-11d2-AA12-08005AD17735}]
"VBClass"="vbsample.SampleActionsManager"
"VBInterface"="{0FC5EC7A-8E00-11D2-AA9A-08005AD17735}"

;-----
; Register drag and drop context menu handlers

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\
IBM.VBSample\shellex\SampleVBFolder\*\
DragDropHandlers\{040606B2-1C19-11d2-AA12-08005AD17735}]
"VBClass"="vbsample.SampleActionsManager"

```

```
"VBInterface"="{0FC5EC7A-8E00-11D2-AA9A-08005AD17735}"

;-----
; Register Drop Handler to accept drops of objects

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.VBSample\
shell\SampleVBFolder\*\
DropHandler]
@="{040606B2-1C19-11d2-AA12-08005AD17735}"
"VBClass"="vbsample.SampleDropTargetManager"
"VBInterface"="{0FC5EC6E-8E00-11D2-AA9A-08005AD17735}"
```

To customize this section for your own plug-ins, follow these steps:

1. Ensure that the class identifier (CLSID) in the entries above always has the following string: {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's interface ID.
4. If your plug-in is not a drop handler for objects, remove the drop-context menu handler and 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 is used in your Visual Basic source to identify when actions are taken on this folder in System i Navigator.
6. In the file that you created 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 for Visual Basic plug-in registry files:

When developing your own plug-ins, you need to define a unique programmatic identifier (ProgID) for your plug-in. You must specify a unique ProgID for use throughout the file.

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 provided. In the sample plug-in, the string IBM.Sample identifies IBM as the vendor, and Sample as the description of the function provided by the plug-in. This is used throughout the registry file. It names the directory where your plug-in resides on both the system and the workstation. Replace every occurrence of IBM.Sample in the registry file with your ProgID.

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

Note: System i 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 CLSID. The registry files that are provided with the programming samples already contain these predefined CLSIDs.

Related concepts

"Primary registry key" on page 74

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

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 System i 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 provided. In the following examples, 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 (ProgID). It is used throughout the registry file when you specify the function your plug-in provides. It also names the directory where your plug-in resides on both the system 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 System i 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"  
"MinimumRISCRRelease"="ANY"  
"ProductID"="NONE"  
"ServerEntryPoint"="NONE"  
"JavaPath"="MsgQueueSample3.jar"  
"JavaMRI"="MsgQueueSample3MRI.jar"
```

Type If the plug-in adds new folders to the System i 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 System i 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 System i Navigator user interface.

MinimumIMPIRelease

A 6-character string that identifies the minimum release of i5/OS running on IMPI hardware that the plug-in requires. The string should be of the form `vvrrmm`, where `vv` is the i5/OS 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 i5/OS 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 i5/OS running on RISC hardware that the plug-in requires. The string should be of the form vvrrmm, where vv is the i5/OS 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 i5/OS 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 System i 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 system, the value of this field should be NONE.

Multiple comma-separated product IDs can 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 System i Navigator when it needs to determine whether the plug-in is supported on a particular system. 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 can 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 - System i 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. System i Navigator searches for each JAR file after first adding a suffix to the name with the appropriate Java language and country identifiers. In the development version of the registry file, this field can contain an empty string because the resources for the base locale (typically US English) should reside in the code JAR.

Defining new folders

```
;-----  
; 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 System i 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 run time.

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

ADF	Application Development folder
AS4	System folder
BKF	Backup folder
BOF	Basic Operations folder
CFG	Configuration and Service folder
DBF	Database folder
FSF	File Systems folder
MCN	Management Central folder
MCS	Management Central Configuration and Service folder
MDF	Management Central Definitions folder
MMN	Management Central Monitors
MST	Management Central Scheduled Tasks
MTA	Management Central Task Activity
MXS	Management Central Extreme Support
NSR	Network Servers folder
NWF	Network folder
SCF	Security folder
UGF	Users and Groups folder
WMF	Work Management 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 System i 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 System i 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 System i 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 System i 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 System i 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 System i 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 System i 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 System i 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\  
  shell\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\  
  shell\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\  
  shell\Sample5*\TaskHandlers\{1827A857-9C20-11d1-96C3-00062912C9B2}]  
"JavaClass"="com.ibm.as400.opnav.MsgQueueSample5.MqTasksManager"  
"JavaClassType"="TasksManager"
```

Supporting drag/drop

```
;-----  
; Register a drop handler for the new folder and its objects  
  
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.MsgQueueSample3\  
  shell\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 shell\ex key. The first identifier in the pair specifies the root folder for a System i 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 a System i 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 can register their intent to add functions to a given object type in the System i Navigator hierarchy. The plug-in should never assume that it is the only system component that is providing functions for a given object type. This applies not only to existing object types, but also

to any new objects that a plug-in might define. If your plug-in is widely used, nothing prevents 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 System i Navigator to support the designated function.

SSL support

If a plug-in's communications with the system are performed by the Sockets API or some other low-level communications service, then it is the responsibility of the plug-in to support SSL if SSL has been requested. If the plug-in does not provide this support, it should specify "Support Level"=dword:00000000. This indicates that the plug-in does not support SSL. When this is done, the plug-in's function is disabled if the user has requested a secure connection.

```
;-----  
; 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 (MFC) Library classes do not support the creation of property pages for a property sheet handler. However, you can use IBM-provided CExtPropertyPage in place of the MFC class CPropertyPage.

Property pages implemented by System i Navigator plug-ins should have subclass CExtPropertyPage. The class declaration can be found in the header file PROEXT.H, and the implementation is contained in the file PROEXT.CPP. Both files are provided as part of the sample plug-in.

Note: It is necessary to include PROEXT.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, System i Navigator automatically adds Properties to the context menu for the object and calls any registered property sheet handlers to add pages to the property sheet if the context menu item is selected.

In certain cases, a plug-in might implement a Properties context menu item that is defined for one of its own object types as a standard Windows dialog instead of as a property sheet. A flag is defined for this situation. It might be returned to System i 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 a user's property sheets, the key that specifies the CLSID of the property sheet handler must specify a PropSheet field. This field identifies the property sheet to which the specified handler will add pages. Here is an example.

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

Valid values for the PropSheet field are:

PropSheet field valid values				
Groups	Personal	Security or capabilities	Jobs	Networks
Groups-Before-All	Personal-Before-All	Capabilities-Before-All	Jobs-Before-All	Networks-Before-All
Groups-After-Info	Personal-After-Name	Capabilities-After-Privileges	Jobs-After-General	Networks-After-Servers
	Personal-After-Location	Capabilities-After-Auditing	Jobs-After-Startup	Networks-After-General
	Personal-After-Mail	Capabilities-Before-Other	Jobs-After-Display	
		Capabilities-After-Other	Jobs-After-Output	
			Jobs-After-International	

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

Restriction: Property sheets for System i user objects currently have this restriction. Multiple property sheet handlers for the various property sheets associated with a system user cannot be implemented on the same implementation class. Each property sheet requires a separate CLSID.

Related concepts

"IA4PropSheetNotify interface specifications listing" on page 30

The IA4PropSheetNotify interface supplies notifications to the implementation of IShellPropSheetExt. These notifications are needed when you add additional property pages to one of the Users and Groups property sheets.

Description of QueryContextMenu flags:

System i Navigator has enhanced its support for the IContextMenu interface.

Ordering of context menu items

System i 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. System i 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.

System i 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 so that you can determine which section of the context menu is 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. (System i 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 System i 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

You cannot use a registry key to specify property pages that are implemented by System i Navigator Visual Basic plug-ins. You must add a specific property page context menu item in your ListManager class to implement a property page. You cannot 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 System i 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:

```
' Returns the attributes of an object in the list.
Public Function ListManager_getAttributes(ByVal item As Object) As Long
    Dim uItem As ItemIdentifier
    Dim nAttributes As ObjectTypeConstants

    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
End Function
```

```
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
```

```
        ' Show the properties
```

```
        frmProperties.Show vbModal
```

```
    End If
```

```
·  
·
```

```
Case Else
```

```
    'Do Nothing
```

End Select

End Sub

Note: The code to create and display the property sheet can be seen in **propsht.frm**

Property sheet handling in Java

You can add property pages to Java plug-in property sheets. Then, you can 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\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.

Example: Java Properties Manager:

This example shows a Java Properties Manager code sample.

Note: By using the code examples, you agree to the terms of the “Code license and disclaimer information” on page 91.

```
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;

}

}

```

Secure Sockets Layer registry entry

System i Navigator users can request a secure connection to a system by selecting the **Use Secure Sockets Layer** checkbox on the Connection tabbed page of the property sheet for System i objects. When this is done, only System i Navigator components that are capable of supporting Secure Sockets Layer (SSL) communications are enabled for activation by the user.

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

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

When a plug-in's communications with the system 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 SSL has been requested. If the plug-in does not provide this support, the plug-in should indicate that it does not support SSL as shown in the following example. When this is done, the plug-in's function is 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.

```
;-----  
; Example registry key that  
; says this plug-in supports SSL  
{HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.Sample\SSL}  
"Support Level"=dword:00000001
```

Code license and disclaimer information

IBM grants you a nonexclusive copyright license to use all programming code examples from which you can generate similar function tailored to your own specific needs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CANNOT BE EXCLUDED, IBM, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS IBM, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. DIRECT, SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS.

SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF DIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, SO SOME OR ALL OF THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

- | The licensed program described in this document and all licensed material available for it are provided
- | by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement,
- | IBM License Agreement for Machine Code, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

This Developing System i Navigator plug-ins publication documents intended Programming Interfaces that allow the customer to write programs to obtain the services of IBM i5/OS.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

i5/OS
IBM
IBM (logo)
iSeries
System i

- | Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks
- | of Adobe Systems Incorporated in the United States, and/or other countries.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of IBM.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.



Printed in USA