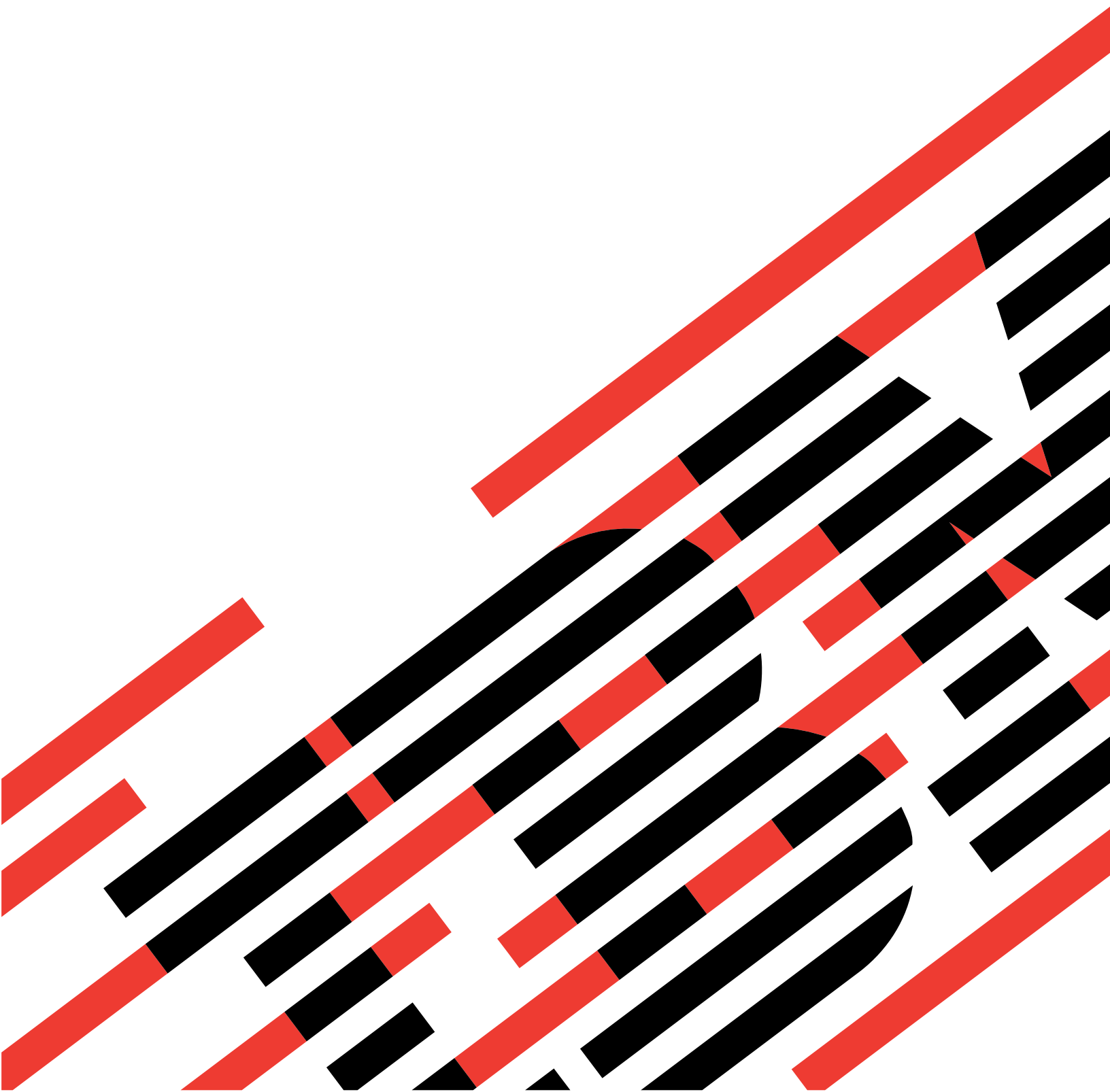




iSeries

Developing iSeries Navigator plug ins





@server

iSeries

Developing iSeries Navigator plug ins

Contents

Developing iSeries Navigator Plug-ins	1
Plug-in support in iSeries Navigator	1
What you can do with a plug-in	2
How plug-ins work	2
Plug-in requirements	4
Distribute plug-ins	5
Identifying plug-ins to iSeries Navigator	11
Install and run sample plug-ins	11
Setting up sample C++ plug-ins	12
Setting up sample Visual Basic plug-ins	13
Setting up the sample Java plug-in	16
Plug-in programming reference	18
iSeries Navigator structure and flow of control for C++ plug-ins.	19
iSeries Navigator COM interfaces for C++	19
iSeries Navigator API listing.	23
Return codes unique to iSeries Navigator APIs	27
iSeries Navigator structure and flow of control for Visual Basic plug-ins.	29
iSeries Navigator Visual Basic interfaces	30
iSeries Navigator structure and flow of control for Java plug-ins	31
Customize the plug-in registry files	32

Developing iSeries Navigator Plug-ins

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

Learn about plug-ins:

Plug-in support for iSeries Navigator

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

Install and run the sample plug-in

The Programmer's Toolkit helps you download and run sample plug-ins. You can use these samples to learn about plug-in support in iSeries Navigator. Also, many developers use these samples as a base for their own modifications.

Develop plug-ins:

Plug-in programming reference

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

Distribute plug-ins

The Selective Setup feature in iSeries Access makes it easy to distribute the plug-in to your end users. Use this section to learn how to identify the new plug-in to iSeries Navigator, and where to install the new plug-in.

Code disclaimer information

This topic contains programming examples.

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

All sample code is provided by IBM for illustrative purposes only. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

All programs contained herein are provided to you "AS IS" without any warranties of any kind. The implied warranties of non-infringement, merchantability, and fitness for a particular purpose are expressly disclaimed.

Plug-in support in iSeries Navigator

iSeries Navigator Plug-in support provides a convenient way to integrate your own functions and applications into a single user interface: iSeries 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 iSeries Navigator provides several important benefits. For example, bundling common system tasks into a single location in iSeries Navigator can dramatically simplify common administration and operation functions. Also, iSeries Navigator's GUI interface ensures that your integrated functions can be completed easily, and with only minimal prerequisite skills.

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

- What you can do with a plug-in
- The new functions you can add with a plug-in
- How plug-ins work
- How plug-ins work by examining an example Java plug-in
- Plug-in requirements
- You can develop plug-ins in C++, VB or Java. This topic describes the specific requirements for each language.
- Distributing plug-ins
- You can easily distribute the new plug-in to your end users by placing it on the managing iSeries server. iSeries Access for Windows Selective Setup then detects the new plug-in and installs it on your client PCs.

What you can do with a plug-in

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

Context menus

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

Property pages

Use property pages to support customized attributes, 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

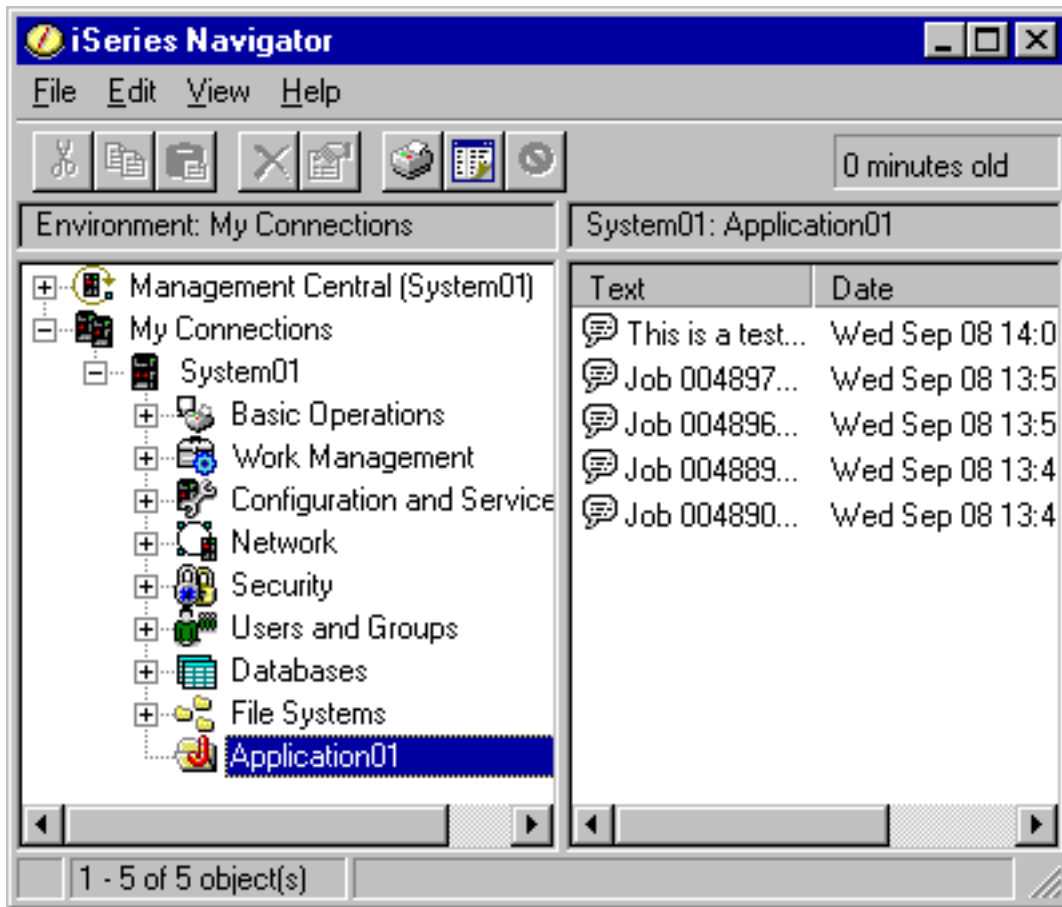
You can add your own customized folders and objects into the iSeries Navigator tree hierarchy.

How plug-ins work

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

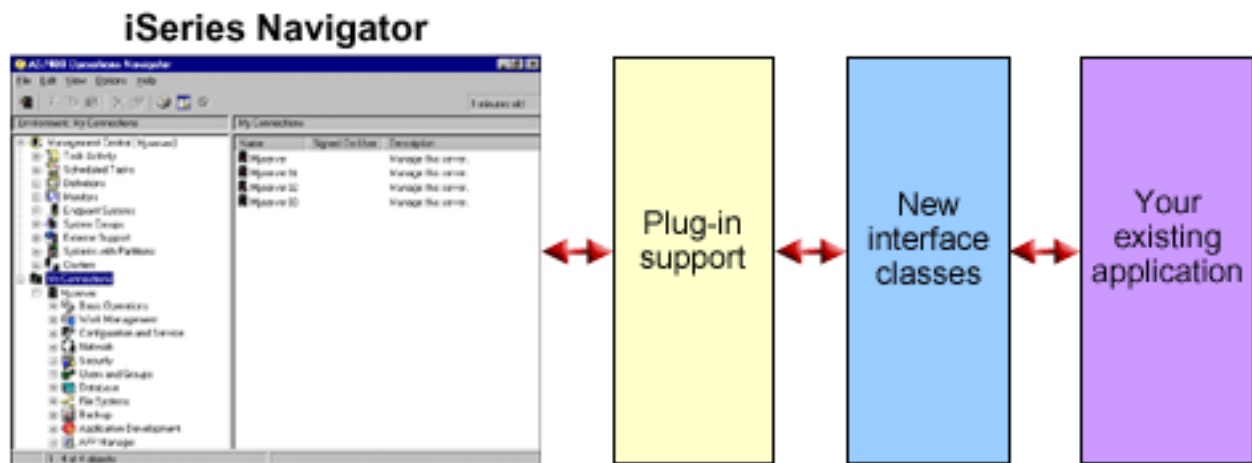
After identifying the new plug-in to the Windows registry, iSeries Navigator will find the new plug-in and install it in a new configuration. Afterwards, the new container will appear in the iSeries Navigator hierarchy. When the user selects the container, the plug-in's Java code is called to obtain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

iSeries Navigator dialog — messages in the message queue



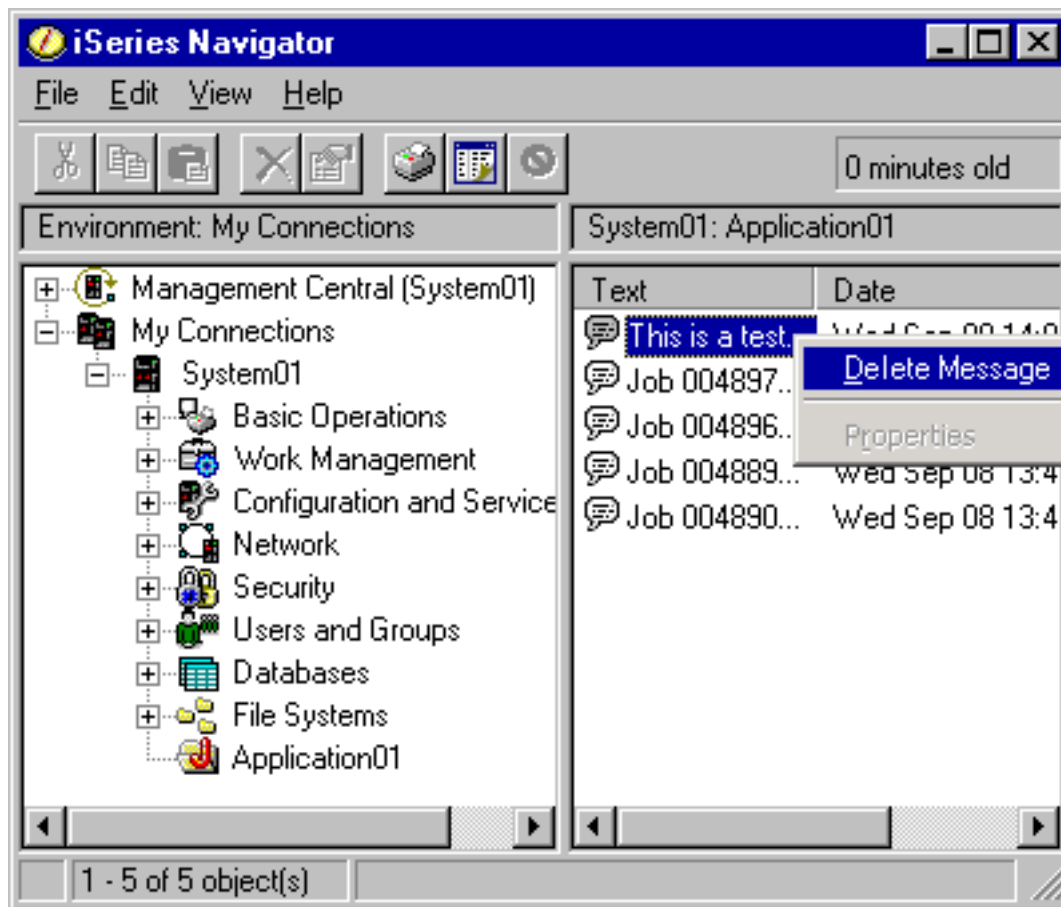
iSeries Navigator communicates with the Java plug-in by calling methods defined on a Java interface: ListManager. This interface lets Java applications supply list data to the Navigator's tree and list views. To integrate your application into iSeries Navigator, you create a new Java class that implements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

How iSeries Navigator calls an application to obtain list data



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

iSeries Navigator object context menu



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

Plug-in requirements

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

C++ plug-ins

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

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

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

Java plug-ins

Java plug-ins run on the IBM runtime for Windows(R), Java Technology Edition. The following table indicates the version of Java installed with iSeries Access for Windows:

Release	JRE	Swing	JavaHelp
V5R2	1.3.1	N/A	1.1.1
V5R1	1.3.0	N/A	1.1.1
V4R5	1.1.8	1.1	N/A
V4R4	1.1.7	1.0.3	N/A

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

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

The sample's user interface was developed by using the Graphical Toolbox for Java, which is a part of the Toolbox for Java component. The Toolbox is an optionally installable component of iSeries Access for Windows. It can be installed with the initial installation of the iSeries Access for Windows product or selectively installed later, by using the iSeries Access for Windows Selective Setup program.

Visual Basic plug-ins

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

Distribute plug-ins

You can deliver your plug-in code to iSeries Navigator users by including it with your OS/400 applications. The installation program for the application writes the plug-in's code binaries, registry file, and translatable resources to a folder in the iSeries server integrated file system. After completing this process, your end users can obtain the plug-in from the iSeries Access for Windows folder (with the help of an iSeries NetServer mapped network drive) using the iSeries Access Selective Setup program. Selective Setup copies your plug-in code to the user's machine, downloads the appropriate translatable resources based on the language settings on the user's PC, and runs the registry file to write your plug-in's registry information to the Windows registry. If you are not initially installed, you can also install plug-ins on the initial install using the custom option.

For this type of plug-in...	Install in this directory...	And include these files...
C++	/QIBM/USERDATA/GUIPLUGIN/ <vendor>.<component> Or: /QIBM/USERDATA/OpNavPlugin/ <vendor>.<component> (To prevent installation without iSeries Access)	<ul style="list-style-type: none"> • The registry file for the plug-in. • The iSeries Access for Windows “iSeries Access for Windows setup file” on page 7 for the plug-in. • The ActiveX server DLL for the plug-in, and any associated code DLLs.
Java	/QIBM/USERDATA/OpNavPlugin/ <vendor>.<component> (Java plug-ins require iSeries Access)	<ul style="list-style-type: none"> • The registry file for the plug-in. • The iSeries Access for Windows “iSeries Access for Windows setup file” on page 7 for the plug-in. • The Java JAR file contains all Java classes, HTML, .gif, PDML, PCML, and serialization files..
Visual Basic	/QIBM/USERDATA/OpNavPlugin/ <vendor>.<component> (VB plug-ins require iSeries Access)	<ul style="list-style-type: none"> • The registry file for the plug-in. • The iSeries Access for Windows “iSeries Access for Windows setup file” on page 7 for the plug-in. • The ActiveX server DLL for the plug-in, and any associated code DLLs.

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

Additionally, all plug-ins must create at least one directory below the <vendor>.<component> subdirectory called MRI29XX, where XX identifies a supported language. 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” on page 11 for the plug-in.

Upgrading or uninstalling the plug-in

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

Restricting access to the plug-in with system policies and Application Administration

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

iSeries Access for Windows setup file

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

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

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

- “Example: Information section of setup.ini”
- “Example: Service section of setup.ini” on page 8
- Sections to “Example: Identify files section of setup.ini” on page 8 to install on the client workstation

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

```
[Plugin Info]
Name=Sample plug-in
NameDLL=sampmri.dll
NameResID=128
Description=Sample plug-in description
DescriptionDLL=sampmri.dll
DescriptionResID=129
Version=0
VendorID=IBM.Sample
SupportExpress=YES
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.
Version	<p>A numeric value that indicates the release level of the plug-in. The iSeries Access for Windows Check Service program uses this value 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 to the current Version value of the installed plug-in that is on the client workstation. When this Version value is greater than the one already existing on the client workstation, the iSeries Access Login Service Check program upgrades the plug-in to the new Version.</p>

Field in [Plugin Info] section of Setup.ini	Description of field
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 iSeries Access registry tree. The VendorID must be identical to the <VENDOR>.<COMPONENT> portion of the path where the plug-in will be installed on the iSeries server.
SupportExpress	SupportExpress is optional. This indicates that the plug-in is supported in iSeries Access for Windows, and that it will function correctly. If SupportExpress is set to NO or doesn't exist, and the user selects to install this plug-in, a dialog box titled iSeries Navigator Plug-in Not Supported will appear. This notifies you that you will be able to install the plug-in, but that it isn't supported in iSeries Access. If you don't want this dialog box to appear every time the plug-in is installed, and you know that the plug-in works with iSeries Access, then add SupportExpress and set it equal to YES.
JavaPlugin	JavaPlugin is used to indicate whether this is a Java plug-in. The install process needs to do some special processing if the plug-in is a Java plug-in. All JAR files must be installed into the \PLUGINS\<VENDOR>.<COMPONENT> directory, and this value is used to determine whether the install process should do this. If the plug-in is a Java plug-in and this value is set to NO or doesn't exist, the plug-in may not work after it is installed.

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

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

Below is a listing of the meaning of each field:

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

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

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

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

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

Note:The 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> (and="" access="" activex="" associated="" client="" code="" directory.="" dll="" dlls)="" for="" here.<br="" install="" normally,="" plug-in="" reside="" server="" the="" under=""></vendor>.<component>> For C++ and Visual Basic , the ActiveX server DLL (and associated code DLLs) for the plug-in reside here. For Java , the Code JAR file name will 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]	Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory that are use counted in the registry and are never removed. These are typically re-distributable files.
[MRI Files]	Files that are copied from the MRI directories of the plug-in on the iSeries server to the CLIENT ACCESS\MRI29XX\ <vendor>.<component> a="" directories="" dll="" for="" include="" is="" locale-dependent="" mri="" name.<="" on="" plug-in="" reside.="" resource="" resources="" td="" the="" this="" typically="" where="" will="" workstation.="" your=""> </vendor>.<component>>
[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 iSeries server to the same directory to which the [Base Files] are installed. This typically is where the JAR MRI29xx resources for the plug-in reside. For each MRI29xx directory supported by the Java plug-in, there needs to be a [Java MRI29xx] section listing those files. This only is used by Java plug-ins.
[Help files]	The .HLP and .CNT files that are copied from the MRI directories of the plug-in on the iSeries server to the CLIENT ACCESS\MRI29XX\ <vendor>.<component> directories="" directory="" files="" hkey_local_machine\software\microsoft\windows\help="" in="" is="" on="" path="" registry.<="" td="" the="" these="" to="" windows="" workstation.="" written=""> </vendor>.<component>>
[Registry files]	The Windows registry file that is associated with the plug-in.

[Dependencies]	<p>Defines the sub components that must be installed before the plug-in can be installed. The values described below are optional. They are only needed if the plug-in requires other sub components to be installed besides the iSeries Navigator base support sub component.</p> <p>Two values are supported: AS400_Operations_Navigator</p> <ul style="list-style-type: none"> This value is used for legacy purposes to identify the sub components that must be installed if the plug-in is installed on iSeries Access V3R2M0. If the plug-in does not support running on iSeries Access V3R2M0, this value should not be specified. The sub components are specified in a comma-delimited list. A single sub component is specified as a single number (AS400_Operations_Navigator=3). The CWBUN.H header file contains a list of constants that are prefixed with CWBUN_OPNAV_. These constants provide the numeric values that are used in the comma-delimited list for AS400_Operations_Navigator. <p>AS400_Client_Access_Express</p> <ul style="list-style-type: none"> This value is used to identify the sub components that must be installed if the plug-in is installed on iSeries Access. The sub components 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 are prefixed with CWBAD_COMP_. These constants provide the numeric values that are used in the comma-delimited list for AS400_Client_Access_Express. There are several CWBAD_COMP_ constants that identify PC5250 font sub components. These constants must not be used in the AS400_Client_Access_Express value and are listed below: <pre>//5250 Display and Printer Emulator sub components #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> <ul style="list-style-type: none"> This value is ignored by Client Access V3R2M0. <p>Note:iSeries Access will use the AS400_Client_Access_Express value if it exists. If it does not exist, it will use the AS400_Operations_Navigator value, if it exists. If neither value exists, then this section is ignored.</p>
[Service Base Files]	Files that are copied to \PLUGINS\<VENDOR>.<COMPONENT> under the iSeries Access install directory.
[Service Shared Files]	Files that are copied to the iSeries Access Shared directory.
[Service System Files]	Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory.
[Service Core Files]	Files that are copied to the \WINDOWS\SYSTEM or \WINNT\SYSTEM32 directory. These files are use counted in the registry, are never removed, and are typically re-distributable files.

[Service Registry Files]	The Windows registry file that is associated with the plug-in.
--------------------------	--

MRI setup file

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

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

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

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

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

Identifying plug-ins to iSeries Navigator

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

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

Install and run 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 iSeries Access Selective Setup.

- Setting up the sample C++ plug-in
Download the sample C++ plug-in and get it running in iSeries Navigator.
- "Setting up sample Visual Basic plug-ins" on page 13
Download the sample Visual Basic plug-in and get it running in iSeries Navigator.



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

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

Setting up sample C++ plug-ins

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

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

Download the C++ plug-in	<p>Download the executable file cppsmppq.exe</p>  <p>. When you run the file it will extract all the files associated with the plug-in. Make a new directory, c:\MyProject, and copy all the files into it. If you create a different directory, you will have to modify registry file to specify the correct location for the plug-in.</p>
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. <p>Note: If the new directory is not c:\MyProject, you will need to change the registry file.</p> <ol style="list-style-type: none"> 2. Copy all of the sample files into this directory. You can download the samples from the Programmer's Toolkit - iSeries Navigator Plug-ins Web page  <p>.</p> <ol style="list-style-type: none"> 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. <p>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.</p>


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 and select Open Workspace... and select the MyProject directory. 2. Specify Makefiles (*.mak) in Files of Type: 3. Select sampmri.mak and click Open. 4. Open the Build menu and select Rebuild All to compile and link the DLL.
Register the ActiveX server .dll	<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 the iSeries Navigator. If you specified a directory other than c:\MyProject, complete the following steps.</p> <ol style="list-style-type: none"> 1. Open the SAMPDBG.REG file in the Developer Studio (or use your chosen text editor). 2. Replace all occurrences of "c:\MyProject\" with "x:\<dir>", where x is the drive letter where your directory resides and <dir> is the name of the directory. 3. Save the file. 4. In Windows Explorer, double-click the SAMPDBG.REG file. This will write the entries in the registry file to the Windows registry on your machine. Note: In Windows NT, you must login with administrative privileges on your workstation to write to the Windows registry.
Run iSeries Navigator in the debugger	<p>To run iSeries Navigator and observe the sample plug-in in action, complete the following 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 iSeries Navigator executable in the iSeries Access Install directory on your workstation. The path will be C:\PROGRAM FILES\IBM\CLIENT ACCESS\CWBUNNAV.EXE or something similar. 3. Click OK. The main window of the iSeries Navigator will open. 4. Because you have just registered a new Navigator plug-in, a dialog in iSeries Navigator will prompt you to scan for the new plug-in. 5. After the progress indicator finishes, click OK in the resulting dialog. 6. After the Navigator window refreshes, a new folder (3rd Party Sample Folder) appears in the hierarchy under the iSeries server that was initially selected. You can now interact with the plug-in in iSeries Navigator and observe its behavior in the debugger.

Setting up sample Visual Basic plug-ins

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

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

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

Download the VB plug-in	<p>Download the executable file vbopnav.exe</p>  <p>. When you run the file it will extract 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 will have to modify registry file to specify the correct location for the plug-in.</p>
-------------------------	--

Create the VB project	Open vbsample.vpb in Visual Basic. In the reference dialog, select IBM iSeries Access for Windows ActiveX Object Library , and iSeries Navigator Visual Basic Plug-in Support . Note: If either of these references do not appear in your References dialog, select Browse and look for cwbx.dll and cwbnvbi.dll in the iSeries Access for Windows shared directory. The IBM iSeries Access ActiveX Object Library contains OLE automation objects that the sample application requires to make remote command calls to the iSeries server. The iSeries 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 doesn't compile and 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 VBSamplewin32 directory. 2. In Files of Type:, 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 iSeries Navigator	Start iSeries Navigator, and click on the "+" next to an iSeries server to expand the tree. iSeries Navigator will detect the changes to the registry, and prompt you to scan the iSeries server in order to verify that it is capable of supporting the new plug-in. After completing the scan, iSeries Navigator will display the new plug-in in the tree hierarchy.

Sample VB plug-in directory of files

The following tables describe all of the files included with the sample VB plug-in for v5r2.

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

VB forms	Description
authority.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

VB Modules	Description
global.bas	Global declarations.

VB 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

VB Binaries	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	Install information for plug-in's translatable resources
setup.ini	Install information for plug-in's executables

Registry entries	Description
vbsmpdbg.reg	Registry file for use during development.
vbsmprls.reg	Registry file for use by iSeries Access 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	iSeries 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.

Setting up the sample Java plug-in

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

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

To set up the sample Java plug-in:

Download the sample Java plug-ins	<p>Download the executable file jvopnav.exe.</p>  <p>When you run this file, it will extract all of the previously mentioned files. You should allow the executable to install the files in the default directory: jvopnav\com\ibm\as400\opnav.</p>
Identify the plug-in to iSeries 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 PC. You must double all back slashes in the path. 4. Save your changes, and double click the registry file.
Run the sample Java plug-in.	<ol style="list-style-type: none"> 1. Start iSeries Navigator, and click on the "+" next to an iSeries server to expand the tree. 2. iSeries Navigator will detect the changes to the registry, and prompt you to scan the iSeries server in order to verify that it is capable of supporting the new plug-in. 3. Click Scan Now 4. iSeries Navigator will scan the iSeries server. When it finishes, it will display a new folder in the hierarchy tree, Java Message Queue Sample 1, 2 or 3. 5. Double click on the new folder 6. The first sample plug-in will display the contents of your default message queue in QUSRSYS on the iSeries server. The second and third samples will display a list of message queues. 7. Add a new message by right-clicking on the message queue folder, and selecting New -> Message. 8. The plug-in displays a PDML dialog allowing you to enter the message text. 9. Delete a message by right-clicking on a message and selecting Delete. You can also do this from the toolbar. 10. If you're using the third sample plug-in, you can select a message, drag it to another queue, and then drop it. 11. The plug-in will then move the message to the other queue.

Sample Java plug-in directory of files

The following tables describe all of the files included with the sample Java plug-ins for v5r2. For more information, read the plug-in's javadoc documentation. These were installed in your jvopnav\com\ibm\as400\opnav\MsgQueueSample1\docs directory. Start with the file Package-com.ibm.as400.opnav.MsgQueueSample1.html.

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

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

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

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

PDML files	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 iSeries 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 Windows. It contains substitution variables that represent the directory path of the iSeries Access for Windows installation directory. When the user invokes the iSeries Access Selective Setup program to install your plug-in from the iSeries server, Selective Setup reads this registry file, fills in the correct directory paths, and writes the entries to the registry on the user's machine. The entries in this file should therefore be kept in sync with the registry file used in development.

Plug-in programming reference

iSeries 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 iSeries Navigator for each type of plug-in, as well as specific reference information regarding the unique interfaces for each language.

C++ Reference

- Flow of Control in iSeries Navigator
- COM Interfaces

- API listing
- Return Codes

VB Reference

- Flow of control in iSeries Navigator
- VB Interfaces

Java Reference

- Flow of control in iSeries Navigator
- Java Classes and Interfaces

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

Plug-in registry files

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

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

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

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

The functions that are implemented by a plug-in logically are grouped into "interfaces." An interface is a set of logically related methods on a class that iSeries Navigator can call to perform a specific function. The Component Object Model supports the definition of interfaces in C++ through the declaration of an abstract class that defines a set of pure virtual functions. Classes that call the interface are known as implementation classes. Implementation classes subclass the abstract class definition and provide C++ code for each of the functions defined on the interface.

A given implementation class may implement as many interfaces as the developer chooses. When creating a new project workspace for an ActiveX server DLL in the Developer Studio, the AppWizard generates macros that facilitate interface implementation. Each interface is declared as a nested class on a containing implementation class. The nested class has no member data and does not use any functions other than those that are defined on its interface. Its methods typically call functions on the implementation class to get and set state data, and to perform the actual work that is defined by the interface specification.

iSeries 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 iSeries Navigator can call to perform a specific function. A plug-in may implement one or more COM interfaces, depending on the type of function that the developer intends to provide. For example, when a user right-clicks an object in the

tree hierarchy, iSeries Navigator constructs a context menu for the object and displays the menu on the screen. The Navigator obtains the menu items by calling each plug-in that has registered its desire to supply context menu items for the selected object type. The plug-ins pass their menu items to the Navigator when it calls their implementation of the **QueryContextMenu** method on the **IContextMenu** interface.

Interface	Method	Description
IContextMenu	QueryContextMenu	Supplies context menu items when a user right-clicks on 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 grayed.
	InvokeCommand	Displays the appropriate dialog and performs the requested action. It's called when the user clicks on 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.
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.

IA4PropSheetNotify, 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 may communicate whether the iSeries user whose properties are being displayed already exists or is being defined, and whether changes should be saved or discarded.

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

See the following topics for more information:

- "Description of IA4HierarchyFolder Interface"
- "IA4HierarchyFolder interface specifications listing"
- "Description of IA4PropSheetNotify interface" on page 23
- "IA4PropSheetNotify interface specifications listing" on page 23

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

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

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

IA4HierarchyFolder interface specifications listing

An item identifier, or 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 with its root at the Desktop.

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

The iSeries Navigator uses item identifiers that adhere to the following given 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.

Link to any of the following IA4HierarchyFolder specifications:

IA4HierarchyFolder::Activate

IA4HierarchyFolder::BindToList

IA4HierarchyFolder::DisplayErrorMessage

IA4HierarchyFolder::GetAttributesOf

IA4HierarchyFolder::GetColumnDataItem

IA4HierarchyFolder::GetColumnInfo

IA4HierarchyFolder::GetIconIndexOf

IA4HierarchyFolder::GetItemCount

IA4HierarchyFolder::GetToolBarInfo

IA4HierarchyFolder::GetListObject

IA4HierarchyFolder::ItemAt

IA4HierarchyFolder::ProcessTerminating

IA4HierarchyFolder::Refresh

Description of IA4PropSheetNotify interface

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

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

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

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

IA4PropSheetNotify interface specifications listing

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

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

Link to any of the following IA4PropSheetNotify interface specifications:

- IA4PropSheetNotify::InformUserState
- IA4PropSheetNotify::ApplyChanges
- IA4PropSheetNotify::GetErrorMessage

iSeries Navigator API listing

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

Function	iSeries Navigator APIs
System values: This API allows the plug-in developer to obtain the current value of an iSeries system value.	cwbUN_GetSystemValue

Function	iSeries Navigator APIs
<p>System handles: These APIs allow the plug-in developer to obtain and to release the current value of an iSeries system object handle that contains connection properties including the secure sockets layer (SSL) settings to be used for the specified iSeries system.</p>	<p>cwbUN_GetSystemHandle</p> <p>cwbUN_ReleaseSystemHandle</p>
<p>User input validation: These APIs allow the plug-in developer to check whether the current user has authority to a particular iSeries object. The APIs also allow the developer to determine if the user has one or more special authorities.</p>	<p>cwbUN_CheckObjectAuthority</p> <p>cwbUN_CheckSpecialAuthority</p>
<p>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 iSeries server.</p>	<p>cwbUN_CheckAS400Name</p>
<p>User profile attributes: This API allows the plug-in developer to obtain the value of any of the user profile attributes for the current iSeries Navigator user.</p>	<p>cwbUN_GetUserAttribute</p>

Function	iSeries Navigator APIs
<p>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.</p>	<p>cwbUN_ConvertPidlToString</p> <p>cwbUN_GetDisplayNameFromItemId</p> <p>cwbUN_GetDisplayNameFromName</p> <p>cwbUN_GetDisplayPathFromName</p> <p>cwbUN_GetIndexFromItemId</p> <p>cwbUN_GetIndexFromName</p> <p>cwbUN_GetIndexFromPidl</p> <p>cwbUN_GetListObject</p> <p>cwbUN_GetParentFolderNameFromName</p> <p>cwbUN_GetParentFolderPathFromName</p> <p>cwbUN_GetParentFolderPidl</p> <p>cwbUN_GetSystemNameFromName</p> <p>cwbUN_GetSystemNameFromPidl</p> <p>cwbUN_GetTypeFromItemId</p> <p>cwbUN_GetTypeFromName</p> <p>cwbUN_GetTypeFromPidl</p>

Function	iSeries Navigator APIs
<p>Refresh the iSeries Navigator window: Following the completion of an operation on behalf of the user, these APIs enable execution of a request by the plug-in to refresh the tree and list views or to place a message in the Navigator status bar.</p>	<p>cwbUN_RefreshAll</p> <p>cwbUN_RefreshList</p> <p>cwbUN_RefreshListItems</p> <p>cwbUN_UpdateStatusBar</p>
<p>ODBC connections: These APIs allow the plug-in developer to reuse and end the handle for an ODBC connection that already has been obtained by the Database component of the iSeries Navigator.</p>	<p>cwbUN_GetODBCConnection</p> <p>cwbUN_EndODBCConnections</p>
<p>Access iSeries Navigator icons: These APIs allow the plug-in developer to access the icon image lists for objects that appear in the Navigator object hierarchy.</p>	<p>cwbUN_GetIconIndex</p> <p>cwbUN_GetSharedImageList</p>
<p>Application Administration: These APIs allow the plug-in developer to programmatically determine whether a user is denied or allowed use of an Administrable function. An Administrable function is any function whose use can be controlled through the Application Administration sub component of iSeries Navigator.</p>	<p>cwbUN_GetAdminValue</p> <p>cwbUN_GetAdminValueEx</p> <p>cwbUN_GetAdminCacheState</p> <p>cwbUN_GetAdminCacheStateEx</p>
<p>Install: This API allows the plug-in developer to determine if an iSeries Navigator sun component is installed.</p>	<p>cwbUN_IsSubcomponentInstalled</p>

Function	iSeries Navigator APIs
<p>Directory Services: These APIs provide information about the Directory Services (LDAP) server on an iSeries computer, and functions to connect to the server. The connection functions enable you to connect to a server using information (distinguished names, password, etc.) cached by the iSeries Access for Windows. The connection functions use the LDAP client shipped with iSeries Access (LDAP.LIB and LDAP.DLL) and therefore require that your application use that client.</p> <p>Functions that use strings are available in ANSI and Unicode versions.</p> <p>Functions that return distinguished names and other strings for use with LDAP client APIs also are provided in a UTF-8 version for use with LDAP version 3 servers.</p>	<p>cwbUN_OpenLocalLdapServer</p> <p>cwbUN_FreeLocalLdapServer</p> <p>cwbUN_GetLdapSvrPort</p> <p>cwbUN_GetLdapSvrSuffixCount</p> <p>cwbUN_GetLdapSvrSuffixName</p> <p>cwbUN_OpenLdapPublishing</p> <p>cwbUN_FreeLdapPublishing</p> <p>cwbUN_GetLdapPublishCount</p> <p>cwbUN_GetLdapPublishType</p> <p>cwbUN_GetLdapPublishServer</p> <p>cwbUN_GetLdapPublishPort</p> <p>cwbUN_GetLdapPublishParentDn</p> <p>cwbUN_OpenLdapBindInfo</p> <p>cwbUN_FreeLdapBindInfo</p> <p>cwbUN_GetLdapServerBindDn</p> <p>cwbUN_BindToLdapServerOnAs400</p> <p>cwbUN_BindToLdapServer</p> <p>cwbUN_NullBindToLdapServerOnAs400</p> <p>cwbUN_NullBindToLdapServer</p>

Return codes unique to iSeries Navigator APIs

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 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 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.

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

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

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

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

- ListManager
- ActionsManager
- DropTargetManager

iSeries Navigator data for Visual Basic plug-ins

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

iSeries Navigator services for Visual Basic plug-ins

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

iSeries Navigator Visual Basic interfaces

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

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

There are three iSeries Navigator interface classes:

- "iSeries Navigator ListManager interface class"
- "iSeries Navigator ActionsManager interface class"
- "iSeries Navigator DropTargetManager interface class" on page 31

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

iSeries Navigator ListManager interface class

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

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

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

iSeries Navigator DropTargetManager interface class

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

iSeries Navigator structure and flow of control for Java plug-ins

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

When a user is interacting with iSeries 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 iSeries Navigator that are generated in response to user actions. For example, when a user right-clicks on an object in the Navigator hierarchy, the Navigator constructs a context menu for the object, and displays the menu on the screen. The Navigator obtains the menu items by calling each plug-in that has registered its intent to supply context menu items for the selected object type.

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

- ListManager
- ActionsManager
- DropTargetManager

Product architecture for iSeries Navigator plug-ins

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

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

iSeries Navigator data for Java plug-ins

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

An iSeries Navigator plug-in sometimes will need to affect the behavior of the main Navigator window. For example, following completion of a user operation, it may be necessary to refresh the Navigator list view or

to insert text into the Navigator's status area. Utility classes are supplied in the package com.ibm.as400.opnav that provide the required services.

Customize the plug-in registry files

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

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

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

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

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

Special considerations for the registry files

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

Customize the C++ registry values

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

Primary registry key

```
; -----  
; 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"="sampext.dll"
```

See the topic Example: Primary registry key for a description of each of the fields and the recommended values.

Data Server Implementation

```
-----  
; This section will register an IA4HierarchyFolder implementation for each new  
; folder added to the iSeries 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 will add more than one new folder to the hierarchy, you must duplicate this section of the registry file for each additional folder, making sure to generate a separate GUID for each folder. If your plug-in doesn't add any folders, you can remove this section.

1. Change the name of the DLL to match the name of the DLL that is generated by your new project workspace.
2. Generate and copy a new GUID (See the global changes section at the bottom of this page)
3. Replace both occurrences of the CLSID in this section of the registry with the new GUID string you just generated.
4. Search for the string "IMPLEMENT_OLECREATE" in your version of the file SAMPDATA.CPP
5. Paste the new GUID over the existing CLSID in the comment line, then change the CLSID in the IMPLEMENT_OLECREATE macro call to match the hex values in your new GUID. Replace the word "Sample" with the name of your new folder.
6. Create two new source files for each new GUID, using a renamed copy of SAMPDATA.H and SAMPDATA.CPP as a base.
- 7.

Note: The header file (.H) contains the class declaration for the new implementation class. The implementation file (.CPP) contains the code that obtains the data for the new folder.

8. Replace all occurrences of the class name "CSampleData" in the two source files with a class name that is meaningful in the context of your plug-in.
9. To add the new implementation files to the project workspace, open the **Insert** menu and select **Files Into Project...**
10. Because you are duplicating SAMPDATA.CPP in this way, all your new folders will initially contain library objects.

Shell plug-in implementation

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

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

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

Shell plug-in implementation for objects

```
;-----  
; Register a context menu handler for the new folder and its objects  
  
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-in\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 iSeries Navigator  
; Auto Refresh function)  
  
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-in\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-in\IBM.Sample\shellex\Sample\  
\DragDropHandlers\{3D7907A1-9080-11d0-82BD-08005AA74F5C}]  
  
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-in\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
```

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

1. Replace the CLSID in this section with the new GUIDs.
2. If your plug-in will not add additional property pages to a property sheet for a folder or object, then remove the registry entry for the property sheet handler.
3. If your plug-in will not be a drop handler for objects, remove the drag and drop context menu handler and drop handler registry entries.
4. Edit the subkeys \Sample*. For more information see, Shell plug-ins.
5. Edit or remove the code in your version of EXTINTFC.CPP, that checks for the object types defined by the sample.
You should see the folders, context menu items, property pages, and drop actions from the sample, depending on how much function from the sample you decided to retain

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

Global changes

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

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

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

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

For your iSeries Navigator C++ plug-in to work properly, you must replace specific CLSIDs in your new registry file with GUIDs that you generate.

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

Follow these steps to generate your GUIDs:

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

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

```
;-----  
; Define the primary registry key for the plugin  
; NOTE: NLS and ServerEntryPoint DLL names must not contain qualified directory paths  
  
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-in\IBM.Sample]  
"Type"="PLUGIN"  
"NLS"="sampmri.dll"  
"NameID"=dword:00000080  
"DescriptionID"=dword:00000081  
"MinimumIMPIRelease"="NONE"  
"MinimumRISCRlease"="030701"  
"ProductID"="NONE"  
"ServerEntryPoint"="sampext.dll"
```

Primary Registry key field	Field Description
Type	If the plug-in adds new folders to the iSeries 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 iSeries 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 iSeries Navigator user interface.
MinimumIMPIRelease	<p>A 6-character string that identifies the minimum release of OS/400 that runs on the IMPI hardware that the plug-in requires. The string should be of the form vvrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 2 Modification Level 0, the value of this field should be "030200."</p> <p>If the plug-in does not support any OS/400 release that runs on IMPI hardware (releases prior to Version 3 Release 6), the value of this field should be "NONE." If the plug-in can support any release that runs on IMPI hardware, the value of this field should be "ANY."</p>
MinimumRISCRelease	<p>A 6-character string that identifies the minimum release of OS/400 that runs on RISC hardware that the plug-in requires. The string should be of the form vvrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 7 Modification Level 1, the value of this field should be "030701."</p> <p>If the plug-in does not support any OS/400 release that runs on RISC hardware (Version 3 Release 6 and above), the value of this field should be "NONE." If the plug-in can support any release that runs on RISC hardware, the value of this field should be "ANY."</p>
ProductID	<p>A 7-character string that specifies the product ID of a prerequisite iSeries server licensed program that is required by the plug-in. If the plug-in does not require that a particular licensed program be installed on the iSeries server, the value of this field should be "NONE."</p> <p>Multiple comma-separated product IDs may be specified if multiple IDs exist for the same product.</p>
ServerEntryPoint	The name of the code DLL that implements the server entry point. This entry point is called by the iSeries Navigator when it needs to determine whether the plug-in is supported on a particular iSeries server. If the plug-in does not implement the entry point, the value of this field should be "NONE." In the development version of the registry file, this may be a fully-qualified pathname.

JavaPath	The classpath string that identifies the location of your plug-in's Java classes. During development of your plug-in, this field might contain the directory paths for the directories where your class files reside. In the production version of the registry file, it should identify your JAR file names relative to the iSeries Access for Windows install path, each preceded by the iSeries Access for Windows substitution variable that represents the install path.
JavaMRI	The base names of the JAR files that contain locale-dependent resources for the plug-in. iSeries 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, iSeries Navigator will expect the MRI for the base locale (usually US English) to reside in the code JAR files.

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

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

Object type identifiers

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

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

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

Checking for object types

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

Customize the VB plug-in registry values

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

Primary registry key

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

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

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

```
[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"
```

Recommended changes:

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

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

Registering a new folder

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

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

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

See Example: New folder registry key for a description of each field, and the possible values.

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

Recommended changes:

1. Change all occurrences of the name "SampleVBFolder" in the registry file to a unique name that will identify your folder object. The name that is specified in the registry file must match the object name that is specified in your ListManager and ActionsManager Visual Basic classes. For the sample plug-in these Visual Basic source files are: **listman.cls** and **actnman.cls**.
2. Change the name "vbsample.SampleListManager" in the VBClass key to match the program identifier name of your ListManager class. For example, if your ActiveX Server DLL is named foo.dll, and your ListManager implementation class is MyListManager, then the program identifier is "foo.MyListManager". This name is case-sensitive.
3. Change the value of the "VBInterface" key to the ListManager implementation class interface ID.

Registering VB plug-in objects

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

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

Property sheets still can be added to your plug-in by using a context menu item. You cannot use a registry key for a property sheet that is the mechanism that is used for a C++ plug-in. Property sheet handlers including the Auto Refresh property sheet handler are not supported for Visual Basic plug-ins.

```

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

[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\
IBM.VBSample\shell\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\shell\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}"

```

Recommended changes:

1. The CLSID in the entries above should always have the following: "{040606B2-1C19-11d2-AA12-08005AD17735}"
2. The "VBClass" key contains the program identifier (ProgID) of the Visual Basic implementation class.
3. The "VBInterface" key contains the Visual Basic implementation class' interface ID.
4. If your plug-in will not be a drop handler for objects, remove the drag and drop context menu handler and drop handler registry entries.

5. Rename the subkeys \SampleVBFolder*\ and use a unique string to identify your folder object. This name is the object type that will be used in your Visual Basic source to identify when actions are taken on this folder in iSeries Navigator.
6. In the file that you created that was based on the ActionsManager interface, edit the code that checks for the object types that are defined by the sample to reflect the name of your new folder object. The sample's ActionsManager interface is located in actnman.cls.

Global changes:

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

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

Note: iSeries 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.

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

```

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

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

```

Primary Registry key field	Field Description
Type	If the plug-in adds new folders to the iSeries 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 iSeries 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 iSeries Navigator user interface.

MinimumIMPIRelease	<p>A 6-character string that identifies the minimum release of OS/400 that runs on the IMPI hardware that the plug-in requires. The string should be of the form vvrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 2 Modification Level 0, the value of this field should be "030200."</p> <p>If the plug-in does not support any OS/400 release that runs on IMPI hardware (releases prior to Version 3 Release 6), the value of this field should be "NONE." If the plug-in can support any release that runs on IMPI hardware, the value of this field should be "ANY."</p>
MinimumRISCRelease	<p>A 6-character string that identifies the minimum release of OS/400 that runs on RISC hardware that the plug-in requires. The string should be of the form vvrmm, where vv is the OS/400 Version, rr is the Release, and mm is the Modification Level. For example, if the plug-in requires Version 3 Release 7 Modification Level 1, the value of this field should be "030701."</p> <p>If the plug-in does not support any OS/400 release that runs on RISC hardware (Version 3 Release 6 and above), the value of this field should be "NONE." If the plug-in can support any release that runs on RISC hardware, the value of this field should be "ANY."</p>
ProductID	<p>A 7-character string that specifies the product ID of a prerequisite iSeries server licensed program that is required by the plug-in. If the plug-in does not require that a particular licensed program be installed on the iSeries server, the value of this field should be "NONE."</p> <p>Multiple comma-separated product IDs may 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 the iSeries Navigator when it needs to determine whether the plug-in is supported on a particular iSeries server. If the plug-in does not implement the entry point, the value of this field should be "NONE." In the development version of the registry file, this may be a fully-qualified pathname.</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 iSeries Access for Windows install path, each preceded by the iSeries Access for Windows substitution variable that represents the install path.</p>
JavaMRI	<p>The base names of the JAR files that contain locale-dependent resources for the plug-in. iSeries 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, iSeries Navigator will expect the MRI for the base locale (usually US English) to reside in the code JAR files.</p>

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

- ADF Application Development folder
- AS4 iSeries server 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
- MMF Multimedia folder
- NSR Network Servers folder
- NWF Network folder
- SCF Security folder
- UGF Users and Groups folder

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

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

```

;-----
; 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. See Parent field values for a listing of possible values.
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	The CLSID of the IA4HierarchyFolder implementation that should be called by the iSeries Navigator to obtain the contents of the folder. For Java plug-ins , the CLSID always should be: 1827A856-9C20-11d1-96C3-00062912C9B2. For Visual Basic plug-ins , the CLSID should always be: 040606B1-1C19-11d2-AA12-08005AD17735}.
JavaClass	The fully-qualified Java class name of the ListManager implementation that should be called by the iSeries 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 iSeries Navigator to obtain the contents of the folder.
VBIInterface	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 iSeries 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 iSeries 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 iSeries 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 iSeries 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.

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

Globally unique identifiers (GUIDs)

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

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

Defining your plug-in's primary attributes:

```
;-----  
; Define the primary registry key for Message Queue Sample 3.  
  
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\IBM.MsgQueueSample3]  
"Type"="PLUGIN"  
"NLS"="MessageQueuesMRI.dll"  
"NameID"=dword:00000001  
"DescriptionID"=dword:00000002  
"MinimumIMPIRelease"="NONE"  
"MinimumRISCRelease"="ANY"  
"ProductID"="NONE"  
"ServerEntryPoint"="NONE"  
"JavaPath"="MsgQueueSample3.jar"  
"JavaMRI"="MsgQueueSample3MRI.jar"
```

Type

If the plug-in adds new folders to the iSeries 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 iSeries 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 iSeries Navigator user interface.

MinimumIMPIRelease

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

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

MinimumRISCRelease

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

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

ProductID

A 7-character string that specifies the product ID of a prerequisite iSeries server licensed program that is required by the plug-in. If the plug-in does not require that a particular licensed program be installed on the iSeries server, the value of this field should be "NONE."

Multiple comma-separated product IDs may be specified if multiple IDs exist for the same product.

ServerEntryPoint

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

JavaPath

The classpath string that identifies the location of your plug-in's Java classes. During development of your plug-in, this field might contain the directory paths for the directories where your class files reside. In the production version of the registry file, it should identify your JAR files. The JAR file names should not be qualified with any directory names - iSeries 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. iSeries Navigator will search for each JAR file after first suffixing the name with the appropriate Java language and country identifiers. In the development version of the registry file this field may contain an empty string, since the resources for the base locale (usually US English) should reside in the code JAR.

Defining new folders:

```
;-----  
; 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 iSeries Navigator hierarchy has a unique logical type. In the example above, the string Sample3 is the type which will be used to identify the currently selected folder when control is passed to your plug-in at runtime.

Parent

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

ADF	Application Development folder
AS4	iSeries server folder
BKF	Backup folder
BOF	Basic Operations folder
CFG	Configuration and Service folder
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
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

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

The second identifier in the pair identifies the specific object type that the plug-in wants to affect. If "*" is specified, the plug-in will be called for the folder type identified in the first identifier, plus all folders and objects which appear in the hierarchy under that folder. Otherwise, a specific type identifier should be specified, and the plug-in will only be called when the user performs an action on an object of that type.

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

CLSIDs

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

JavaClass

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

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

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

Support Level

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

Property pages for a property sheet handler

The Microsoft Foundation Class Library classes cannot be used to construct property pages for a property sheet handler. However, IBM provides **CExtPropertyPage**, which may be used in place of the MFC class CPropertyPage. Property pages implemented by iSeries Navigator plug-ins should subclass CExtPropertyPage. The class declaration may 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, the Navigator automatically will add Properties to the context menu for the object. Also, when this flag is on, Navigator will call any registered property sheet handlers to add pages to the property sheet when the context menu item is selected.

In certain cases a plug-in may desire to implement a Properties context menu item that is defined for one of its own object types as a standard Windows dialog instead of a property sheet. A flag is defined for this situation that may be returned to the Navigator on calls to IContextMenu::QueryContextMenu. If the flag is returned, no automatic processing for Properties is performed, and it is up to the plug-in to add the context menu item and implement the associated dialog. This flag is documented in "Description of QueryContextMenu flags" on page 50.

If a plug-in intends to add property pages to one of the property sheets for an iSeries user, the key that specifies the CLSID of the property sheet handler must specify a PropSheet field that identifies the property sheet to which the specified handler will add pages. An example follows.

```
;----- ;  
Register a property sheet handler for the Network property sheet for iSeries users  
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY plug-in\IBM.Sample\shell\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	
		Capabilities-After-Other	Jobs-After-International	

To add pages to a property sheet for an iSeries user, the plug-in must implement the IA4PropSheetNotify interface (see "IA4PropSheetNotify interface specifications listing" on page 23).

Restriction:

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

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

Description of QueryContextMenu flags: iSeries Navigator supports the following enhancements to the IContextMenu interface:

Ordering of context menu items

The iSeries Navigator has extended the IContextMenu interface to obtain more precise control over the order in which menu items are added to the menu for a particular folder or object. The Navigator structures its context menus in three sections. This structure ensures that when more than one component adds items to the context menu for an object, the items will still appear in the correct order that is defined for the Windows user interface.

The first section contains actions which are specific to the object type, such as Reorganize for a database table. The second section contains "object creation" items; these items are object types which cascade off of a New menu item. Lastly there are the so-called "standard" Windows menu items, such as Delete or Properties. You may choose to add menu items to any section of the context menu.

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

UNITY_CMF_CUSTOM

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

UNITY_CMF_NEW

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

UNITY_CMF_STANDARD

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

UNITY_CMF_FILEMENU

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

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

Unique property dialogs

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

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

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

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

In the Visual Basic Sample plug-in, a property page is supported for Libraries in the iSeries 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

    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

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

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

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

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

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

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

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

For more information, see the `Properties Manager` example.

Secure Sockets Layer (SSL) registry entry

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

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

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

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

Example: Adding a registry key to enable SSL

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

Note: "Support Level"=dword:00000001 = supports SSL, and "Support Level"=dword:00000000 = does NOT support SSL.

```
-----  
; 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
```




Printed in U.S.A.