Lab: Extending Operations Navigator -- Plugging in Your Applications

Student Exercises

Ali Nelson, IBM Rochester

Session 21LA (403967)

Lab: Extending Operations Navigator -- Plugging In Your Application

Introduction

This lab explains the steps necessary to build an Operations Navigator plug-in application. This lab will give you the opportunity to build a Java GUI plug-in that gets and retrieves data from the iSeries.

Overview and Background Information

Operations Navigator

Operations Navigator (Op Nav) is the iSeries Graphical User Interface. Operations Navigator allows the user to perform many of the iSeries administration functions. Some of these functions include creating users and groups, working with your database, setting up security, administering your TCP/IP configuration, working with a network of iSeries, plus much, much more.

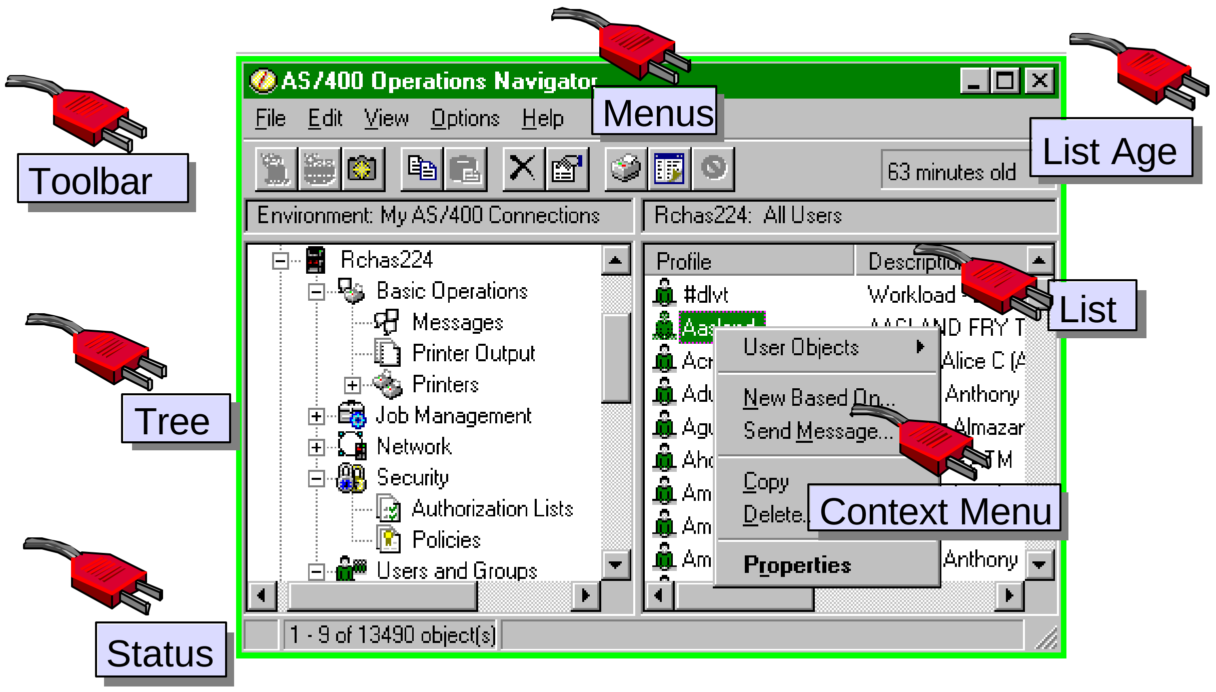
Operations Navigator is shipped with Client Access at no additional charge ("free" with OS/400).

Plug-in Support

You can integrate your own application into Operations Navigator using the plug-in support.
Operations Navigator plug-in supports allows you to build your application using either Visual Basic, C++, or Java. However, Java is the preferred programming language and will be the focus of this lab.

A plug-in application is a set of resources, classes, interfaces, and other "cross-references" that extend Operations Navigator without changing Op Nav's underlying code. The plug-in application developer (You) can build an application and integrate it into Operations Navigator without knowing the internal infrastructure of Op Nav.

The plug-in support gives you a few options to seamlessly integrate your application into Operations Navigator:
•Add your function to an existing context menu, menubar, or toolbar.
•Add a property page to an existing Properties dialog.
•Add your own folder to the tree within an existing folder.



In addition, plug-in support offers a set of common services that your plug-in can take advantage of, including:
•Auto-refresh -- setting up a timed refresh for iSeries objects in the Operations Navigator list view.
•Setting status bar text
•Displaying flyover help

The Java™ Language

Java is a simple, object-oriented, network-aware, portable, interpreted, robust, secure, architecture neutral, high-performance, multithreaded, dynamic language. Java is object-oriented from the ground up. Java organizes code into a collection of classes. Each class is made up of methods and data. Classes can be grouped together and placed in packages.

●**Packages** are similar to iSeries ILE RPG service programs. They enable you to divide your pieces into easily reused units. Packages are Java language constructs. They may contain multiple classes.

●**Classes** are similar to iSeries ILE RPG modules. They enable you to divide your source code into functions ("methods" in Java, procedures and subroutines in RPG) and the variables those functions need. Classes are typically self-contained groupings. They normally contain multiple fields (variables) and methods.

●**Methods** are similar to iSeries ILE RPG procedures and subroutines. They contain all the actual code your program will run. In Java, unlike RPG, executable code can only exist in methods, and methods can only exist inside classes.

Toolbox for java

Introduction

The Toolbox for java is a package of tools, APIs, and predefined GUIs for you to use when building your plug-in application. Your plug-in application can access iSeries data and resources using the Toolbox for java. In addition, you can use the GUI Builder, packaged in the Toolbox, to build your GUI front-end.

The Toolbox for java contains the infrastructure to access the following iSeries data and resources:

- JDBC and record-level access to DB2/400 data
- print resources
- integrated file system
- data queues
- program calls
- command calls
- user lists
- job lists
- job logs
- message queues
- *and many others!*

The Toolbox for java contains the following tools to help you build your GUI:

- GUI Builder
- Resource Converter

The GUI Builder is a visual WYSIWYG editor for building your Java panels. It generates PDML, a simple tag language that was developed by IBM. In addition, you can use the GUI Builder to generate skeleton files for the data beans, event handlers, and button handlers for your GUI panels. Plus, the GUI Builder will also generate skeleton files for the on-line help for your application.

The Resource Converter takes a Windows resource file and converts it to a PDML file. After converting the file to PDML, you can use the GUI Builder to edit and update the GUI panels.

All of the classes in the TOOLBOX FOR JAVA are 100% Pure Java.

The current version of the Toolbox is 'mod 4' which shipped with iSeries version 5 release 1. It runs on any JVM that supports JDK 1.3 or later.

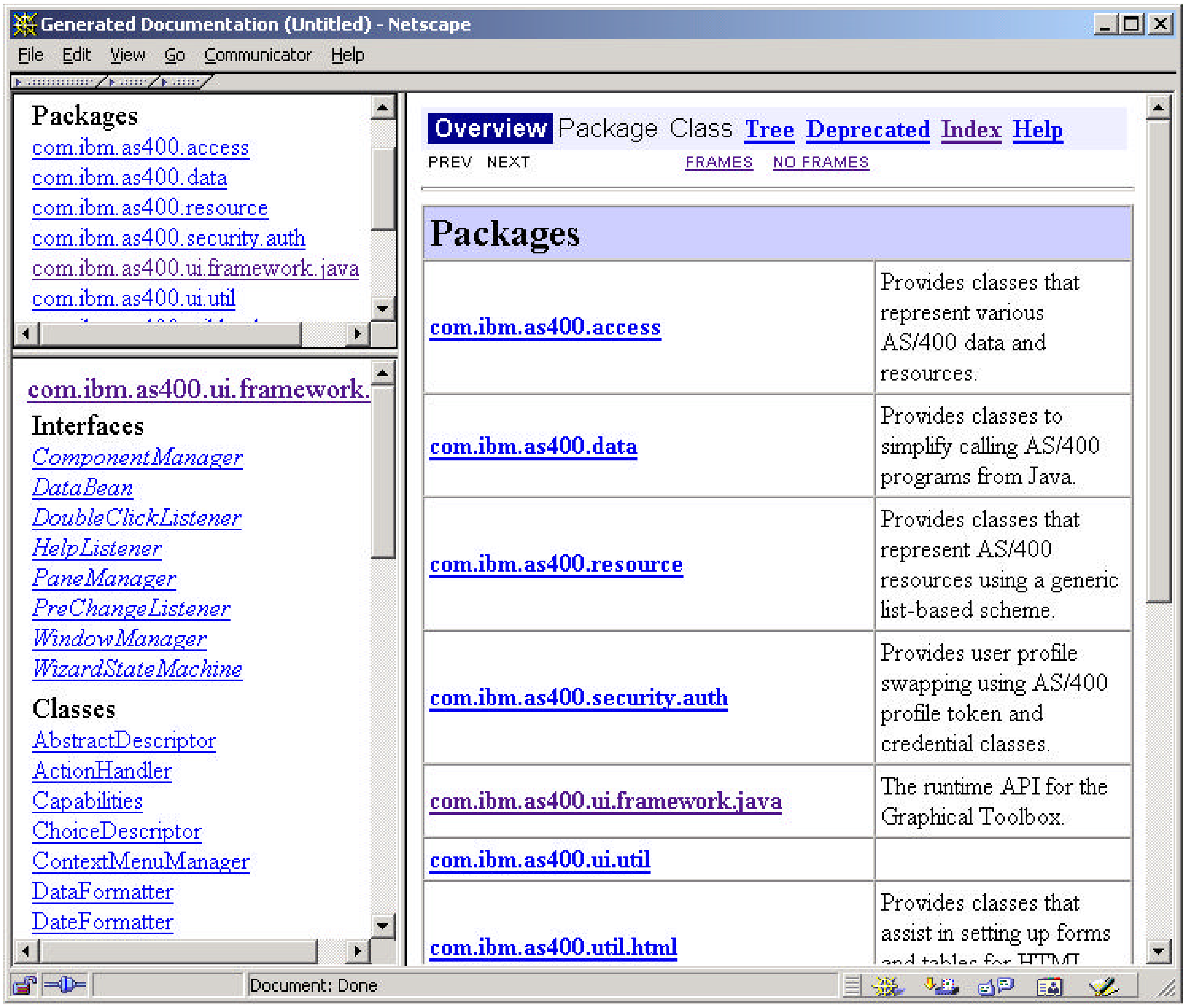
Toolbox 'mod 4' is currently available for download. In addition to new functionality, this version of the Toolbox support Java 2 as well as JDK 1.1.x.

See ***http://www.ibm.com/eserver/iseries/toolbox*** for more information.

Javadoc

The TOOLBOX FOR JAVA provides detailed API information in a Javadoc. Javadoc includes an overview of the class, public constants, and a detailed explanation of each public constructor and public method. As a component is developed, the developers include information in the Java source files. A tool that comes with the JDK pulls out this information and formats it as HTML. An example follows.

The Toolbox Javadoc is installed on the lab PCs. To view it, start Netscape then open file **c:\plugins\V5R1_JavaDoc_PDML/index.html.**


Generated Documentation (Untitled) - Netscape
File Edit View Go Communicator Help
Document: Done

Packages
com.ibm.as400.access
com.ibm.as400.data
com.ibm.as400.resource
com.ibm.as400.security.auth
com.ibm.as400.ui.framework.java
com.ibm.as400.ui.util

com.ibm.as400.ui.framework.

Interfaces
ComponentManager
DataBean
DoubleClickListener
HelpListener
PaneManager
PreChangeListener
WindowManager
WizardStateMachine

Classes
AbstractDescriptor
ActionHandler
Capabilities
ChoiceDescriptor
ContextMenuManager
DataFormatter
DateFormatter

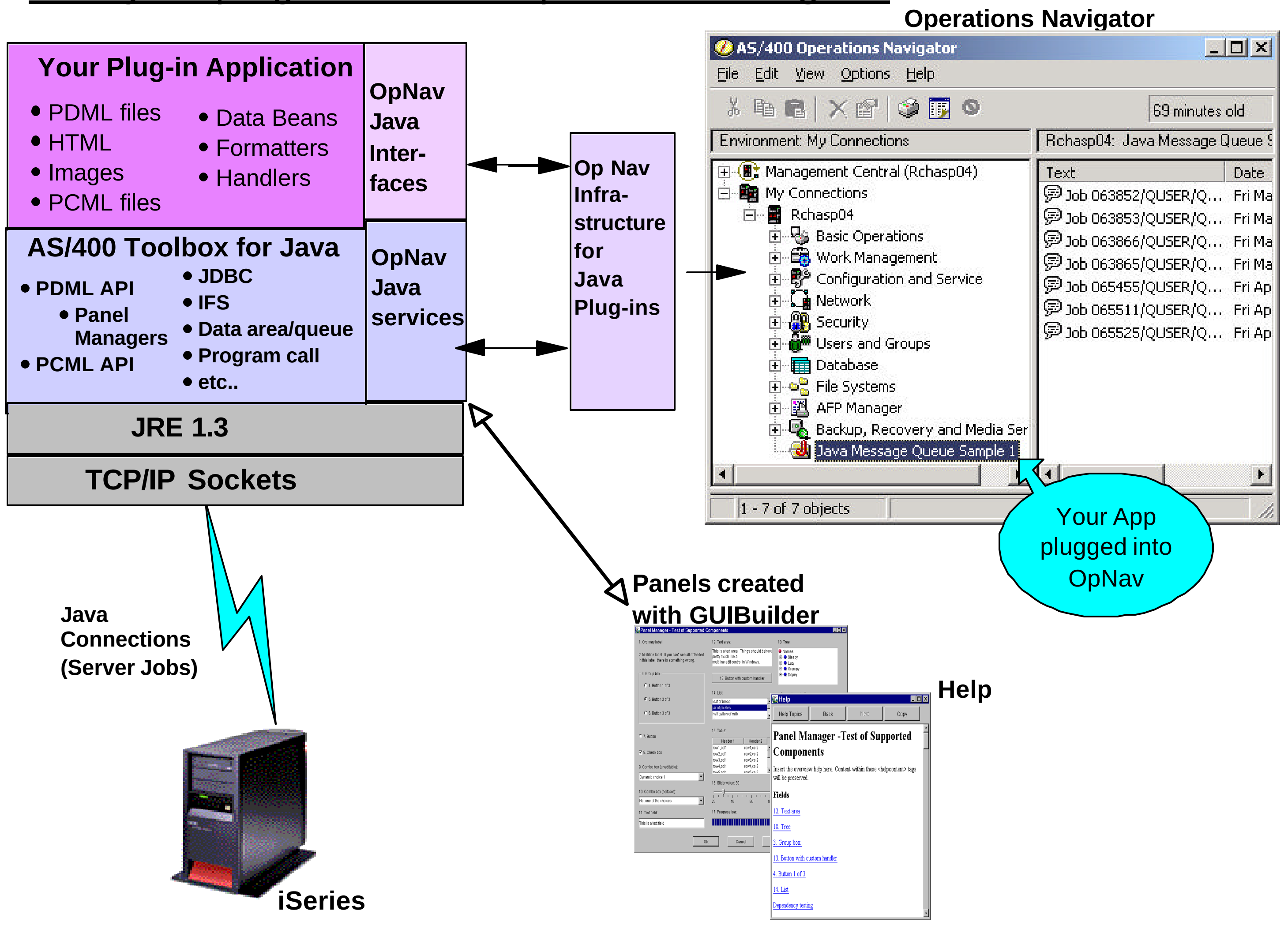
Overview Package Class Tree Deprecated Index Help
PREV NEXT FRAMES NO FRAMES

Packages	
com.ibm.as400.access	Provides classes that represent various AS/400 data and resources.
com.ibm.as400.data	Provides classes to simplify calling AS/400 programs from Java.
com.ibm.as400.resource	Provides classes that represent AS/400 resources using a generic list-based scheme.
com.ibm.as400.security.auth	Provides user profile swapping using AS/400 profile token and credential classes.
com.ibm.as400.ui.framework.java	The runtime API for the Graphical Toolbox.
com.ibm.as400.ui.util	
com.ibm.as400.util.html	Provides classes that assist in setting up forms

The Big Picture: Putting it all Together

The following diagram illustrates how all of these features work together to build an application that is integrated into the iSeries GUI, Operations Navigator.

How your plug-in fits with Operations Navigator



```
Management Central (Rchasp04)
My Connections
  Rchasp04
    Basic Operations
    Work Management
    Configuration and Service
    Network
    Security
    Users and Groups
    Database
    File Systems
    AFP Manager
    Backup, Recovery and Media Ser
    Java Message Queue Sample 1
```

Text	Date
Job 063852/QUSER/Q...	Fri Ma
Job 063853/QUSER/Q...	Fri Ma
Job 063866/QUSER/Q...	Fri Ma
Job 063865/QUSER/Q...	Fri Ma
Job 065455/QUSER/Q...	Fri Ap
Job 065511/QUSER/Q...	Fri Ap
Job 065525/QUSER/Q...	Fri Ap

The Lab

Overview

This lab consists of exercises that demonstrate how to build an Operations Navigator plug-in application from start to finish. In the exercises, you will create a Java GUI using the GUI Builder in the TOOLBOX FOR JAVA. Then, you will plug the GUI into Operations Navigator. In addition, you will will write the code that retrieves and sets the data on the iSeries.

Each exercise gives you some experience with the different steps in the process of building a plug-in application. In most cases, you will start with an existing source file and modify it slightly. If you want to skip some of the exercises, the source code is installed on the lab PCs.

You need the following hardware to complete this lab:

Server - iSeries - OS/400 V4R2 or later

Client - PC - Intel based; 128 MB Memory; 200 MB hard drive
Windows 95, Windows 98, Windows ME, Windows NT or Windows 2000

You need the following software on the client to complete this lab:

- Java Developers Kit 1.3
- TOOLBOX FOR JAVA (Mod 4 -- V5R1)
- Client Access Express V5R1 (with Operations Navigator installed)

You must have the following files (with their appropriate directories) in your CLASSPATH and PATH environment variables:

```
ClassPath= .;C:\java;C:\jdk1.3\lib\classes.zip;
      C:\Program Files\IBM\ClientAccess\jt400\lib\uitools.jar;
      C:\Program Files\IBM\Client Access\classes\jopnav.jar;
      C:\Program Files\IBM\Client Access\jt400\lib\jt400.zip;
      C:\Program Files\IBM\Client Access\jt400\lib\jui400.jar;
      C:\Program Files\IBM\Client Access\jt400\lib\data400.jar;
      C:\Program Files\IBM\Client Access\jt400\lib\util400.jar;
      C:\Program Files\IBM\Client Access\jt400\lib\x4j400.jar;
 C:\Program Files\IBM\Client Access\JRE\Lib\jhall.jar;
 Path=...;C:\jdk1.3\bin;
```

Lab Flow

Most of the exercises include Java source code with a few holes. What is missing is the code that demonstrates a particular feature of the plug-in support. You will fill in this code then compile and run the program. Hints are given in the lab instructions to help you write the code. If you get stuck, the lab solutions are at the end of this handout or the completed source code is available.

Lab Setup

During this lab you will need an iSeries, a userid and password. This information will be given to you by the instructor. Please fill in this information below so that you have it for reference during the lab.

My iSeries **system** is: ______________________________

My iSeries **userid** is: ______________________________

My iSeries **password** is: ______________________________

The PC **source code directory** is C:\plugins\

Tips to Remember

- Java is case-sensitive. Enter all lines exactly in the case they appear in the prototypes that follow.

You are now ready to begin the exercises.

Exercise 1: GUI Builder -- Simple GUI Panel

Introduction

In this exercise, you will create the GUI for the plug-in using the GUI Builder. The GUI Builder is packaged and installed with the TOOLBOX FOR JAVA. The GUI Builder creates Java panels using PDML (Panel Definition Markup Language), a tag language (similar to HTML or SGML) that was created by IBM using XML (Extendible Markup Language).

In addition to using the GUI Builder to create your GUI panels, property sheets, and wizards, you can use the GUI Builder to generate the databeans and the help skeletons for the GUI. Databeans are used in Java to get and set data on the GUI panels. In this exercise, you will generate the databean skeleton files and the help skeleton files for the GUI panel.

The solutions to this lab are in c:\plugins\solutions\Exercise 1 -- GUI Builder\.

Goals of this exercise

At the end of this exercise, you should be able to:

1. Start the GUI Builder.
2. Create a PDML file.
3. Create a GUI panel for your plug-in.
4. Bind user interface objects to a databean.
5. Save the PDML file, generate Java code for the databeans, and generate HTML help for the fields.

Part 1: Start the GUI Builder

1. Start a DOS prompt using the Start menu: select “Programs”, “Command Prompt”.

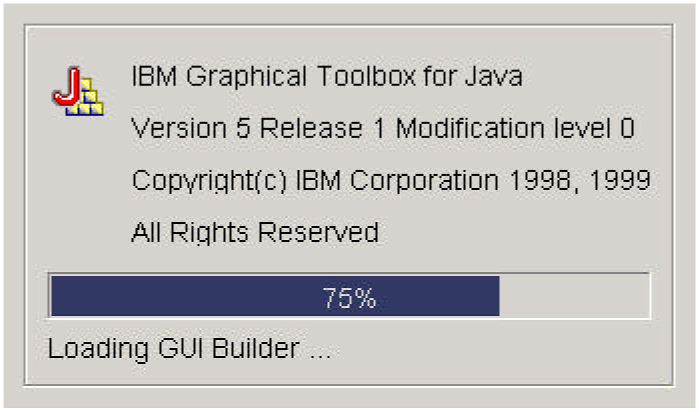
2. Change the current directory to the directory which contains the lab source code (c:\lab25lf). To change directory, enter: **cd C:\plugins**

3. Run batch file **`javasetup`**. This will set up the classpath environment variable to point to the Toolbox jar files. You must run this batch file any time you open a new MS-DOS prompt.

4. Run the GUI Builder. In the DOS prompt, type:

   java com.ibm.as400.ui.tools.GUIBuilder

The following splash screen will appear while your PC starts the GUI Builder.

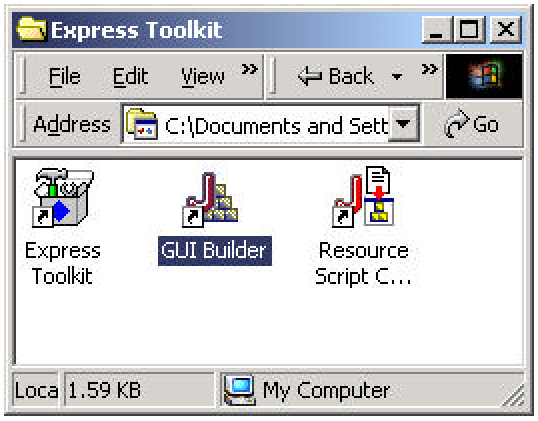


IBM Graphical Toolbox for Java
Version 5 Release 1 Modification level 0
Copyright(c) IBM Corporation 1998, 1999
All Rights Reserved

75%

Loading GUI Builder ...

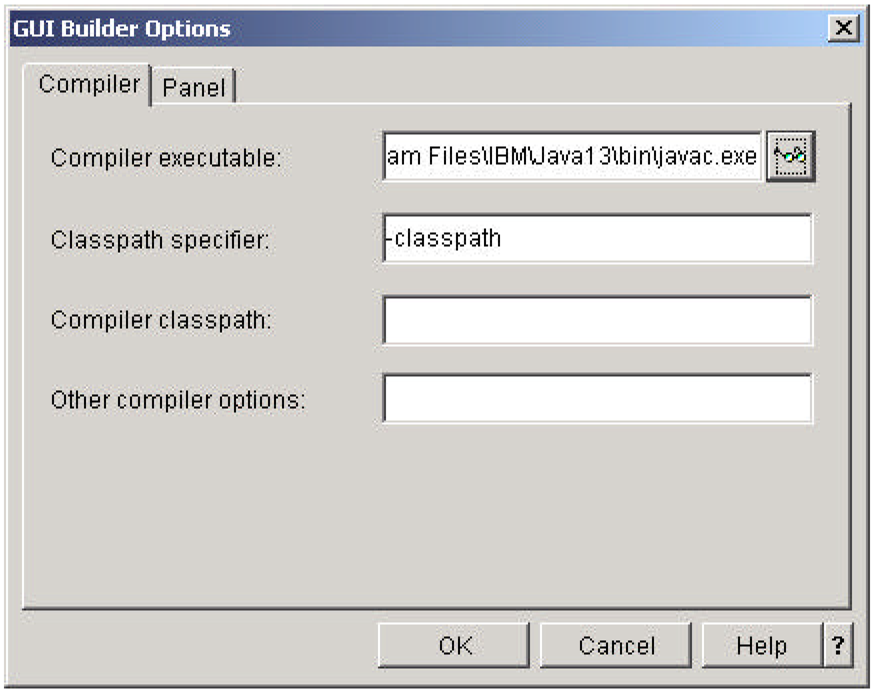
Note: You can also launch the GUIBuilder from the Express Toolkit. You must have installed the Express Toolkit from Client Access selective install. Go into Client Access Express, select the Express Toolkit and then select GUIBuilder.



5. Specify the compiler for the GUI Builder.

The first time that you run the GUI Builder, the Options window will appear. You need to fill in the path for the Java compiler that you wish to use. The GUI Builder will use this when compiling the data beans or resource files that it creates.

For this lab, you will find the java compiler in:
C:\Program Files\IBM\Java13\bin\javac.exe



Compiler executable: am Files\IBM\Java13\bin\javac.exe

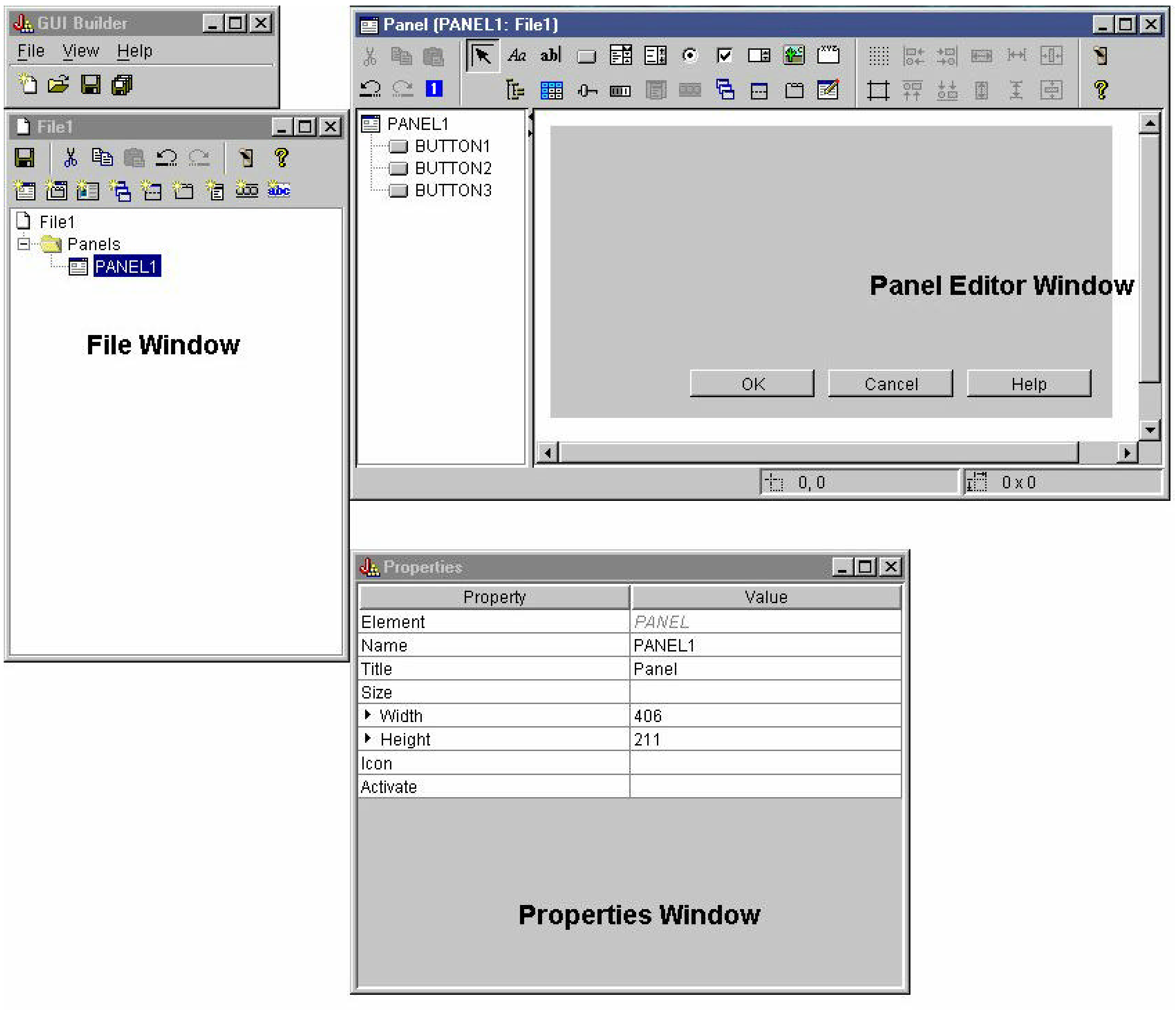
Classpath specifier: -classpath

Compiler classpath:

Other compiler options:

Didn't work? Now what?
- Check the classpath. Does it include everything that was listed in the setup portion of the lab?
- Did you run the javasetup batch file before you started the GUI Builder?

There are 3 main windows that you work with in the GUI Builder: the File window, the Properties window, and the panel editor window.



Property	Value
Element	*PANEL*
Name	PANEL1
Title	Panel
Size	
▸ Width	406
▸ Height	211
Icon	
Activate	

Use the File window to create new PDML files, insert panels, property sheets, wizards, etc., and save the PDML file.   You can cut, copy, and paste the PDML files, or panels within the files from within the File window. You can also setup your user preferences using the menu option **View --> Options** in the File window.

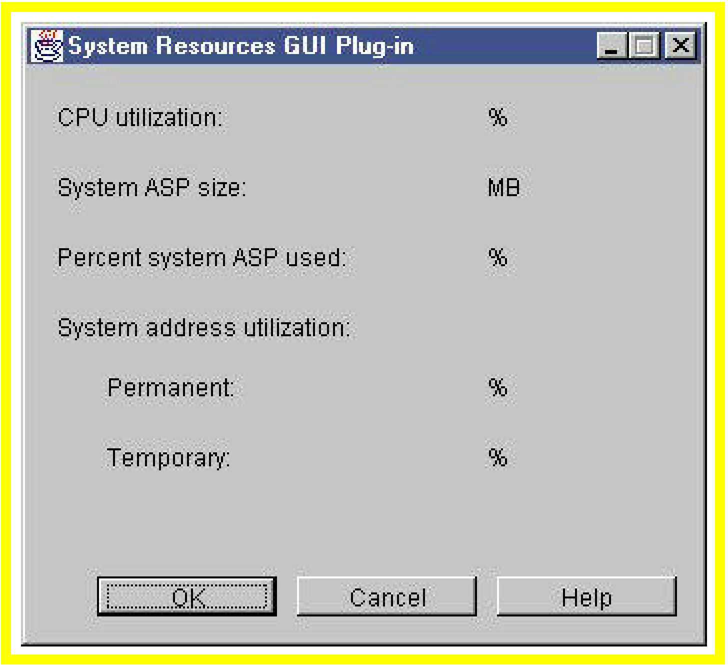
Use the Properties window to change the property of any of the objects in the GUI Builder. For instance, to specify whether you want to generate databeans when the file is saved (a property of the file), you can click on the name of the file in the File window and change the "Generate Databeans" property in the Properties window.

You will use the panel editor window when you create a new panel or edit an existing panel. The panel editor is opened automatically for you when you create a new panel. To edit an existing panel, just double click on the panel name in the File window. The panel editor window includes a setup of user interface objects in the toolbar for that window. You can add these objects to your GUI panel by clicking on the toolbar button and then clicking on the panel. If you are searching for a tool or a user interface object, remember that these toolbar buttons have hover help.

Some tips to remember when using the GUI Builder...
•Press F1 to launch the on-line help. This is especially useful in the properties window.
•Double click on the Properties window to edit the properties.
•Right click on objects to get their context menus.
•Use the Preview function in the panel editor to verify the layout and behavior of the panels.

Remember to ask a lab attendant for help or consult Appendix A for the solutions at any time during this lab, if you get stuck.

Part 2: Create a GUI Panel



CPU utilization: %

System ASP size: MB

Percent system ASP used: %

System address utilization:

Permanent: %

Temporary: %

1. Create a new PDML file. Select the menu option **File --> New** or click on the **New File** toolbar button in the File window.

> **Note:** For this lab, we will only create one panel within one PDML file. However, a PDML file can contain many panels.

2. Insert a new GUI panel. Select the menu option **Insert --> Panel** or click on the **Insert Panel** toolbar button in the File window.

3. Change the Title and Name of the Panel in the Properties window.
   Name: SystemResources
   Title: System Resources GUI Plug-in

4. Add the text labels to the panel in the panel editor. Go to the menu bar and select Label. Drop the label on the panel.

5. Specify the following names and label text in the Properties window:

Name	**Label**
CPU_Label	CPU utilization:
ASPSize_Label	System ASP size:
ASPUsed_Label	Percent system ASP used:
Addresses_Label	System address utilization:
Perm_Label	Permanent:
Temp_Label	Temporary:

2. Align the labels using either the grid tool or the alignment tools in the panel editor toolbar. You can also use the equalize space tools in the toolbar to layout the fields.

Tip: Use the Control (CTRL) key with the mouse to select multiple objects in the panel. For the alignment tools, the last selected item (the item highlighted in blue), is the object that is used as the base for the alignment.

3. Add the labels to display the information from the databean (the information that will be retrieved from the iSeries). In addition to setting the name and label in the Properties window, you should also set the Data Class and Attribute properties for the databeans for these fields. Specify the following information in the Properties window:

Name	Text Label	Data Class	Attribute
CPU_Value	%	SystemStatusEngine	CpuUtilization
ASPSize_Value	MB	SystemStatusEngine	SystemASPSize
ASPUsed_Value	%	SystemStatusEngine	SystemASPPercent
Perm_Value	%	SystemStatusEngine	PermanentAddressesUsed
Temp_Value	%	SystemStatusEngine	TemporaryAddressesUsed

The Data Class and Attribute properties are used in the databean to get and set the values for the label.

Note: Since our GUI is displaying read-only information, you should add labels to display the text from the iSeries. If the GUI required input, you can add any of the other controls (e.g. text field, radio buttons, checkbox, etc.).

4. Align the labels using either the grid tool or the alignment tools in the panel editor toolbar. Preview the panel to verify its layout and content.
5. Set up the file properties to generate databeans and generate help. Click on the file in the File window and change its properties. Or right click on the file name and select the option to Generate Databeans and Generate Help.
6. Save the PDML file. Select the menu option **File --> Save** or click on the **Save** toolbar button. Save the file as SystemStatus.pdml in the plugins directory.
7. Exit the GUI Builder application.

The following files will be saved in your directory.

File name	Contents
SystemStatus.pdml	The PDML definitions including the main panel, layout, and types of user interface objects and their bindings to the databeans.
SystemStatus.html	A skeleton HTML help file for the panel SystemResources. One help file will be generated per panel. This file includes a section for overview help for the panel as well as context sensitive help for the objects on the panel.
SystemStatus.properties	The file stores all of the text that appears on the panels.
SystemStatusEngine.java	The skeleton databean that was generated based on the Data Class information that was entered. It will contain Java code that can be updated to manipulate the data on the panels. One databean file will be created for each Data Class that is specified in Properties.
SystemStatusEngine.class	A compiled version of the SystemStatusEngine.java file.

Didn't work? Now what?
•Did you specify the Data Class and Attribute properties for the labels?
•Verify that you specified to generate databeans and help. Does "Generate Databean" and "Generate Help" properties of the file have a value of "True"?

There are a couple of other File properties that can be set before you save the PDML file:

- **Properties Resource Bundle** vs. **List Resource Bundle** -- The type of resource bundle affects how the text on the panels will be stored. A Properties Resource Bundle stores the panel text in a .properties file. A List Resource Bundle stores the panel text in a .java file and then compiles the .java file to create a .class file. Because the text is compiled, there is quicker access to the text from your GUI. To increase your run-time performance, you should change the PDML file to create a List Resource Bundle. However, if you are in the process of editing and previewing many panels, you should use the Properties Resource Bundle to speed up the return time of the GUI Builder (if you are using the List Resource Bundle, the GUI Builder must compile the panel text everytime you want to Preview the panel). If you change the file's properties to use a List Resource Bundle, you will get the files SystemStatus.java and SystemStatus.class rather than SystemStatus.properties.

- **Serialize** -- To increase the run-time performance of your GUI, you can serialize your PDML file. Each panel in the PDML file is constructed using the serialized file. The IBM XML parser is not used at run-time to contruct the panel, increasing your run-time performance. If you serialize the PDML file, you will get an additional file for each of your panels: SystemResources.pdml.ser.

Exercise 2: Data Beans

Introduction

In this exercise, you will modify the data beans that you generated in the GUI Builder (from Exercise 1) to get and set data on your GUI plug-in panel. The file SystemStatusEngine.java will be modified to retrieve data from the iSeries system using an TOOLBOX FOR JAVA class. After retrieving the data from the iSeries, you will then modify the databean to display this data on the GUI panel.

The GUI Builder creates a skeleton file for the databean that includes the following methods:

- *getAttributeName*() method for each Attribute that you specified -- gets the databean attributes that the user has inputted (e.g. Text field, radio buttons, or checkbox). These methods will not be updated for this exercise since the user can not change any of the data on the GUI.
- *getCapabilities*() method
- *verifyChanges*() method
- *save*() method
- *load*() method -- retrieves the iSeries data and sets the databean attributes

The solutions to this lab are in c:\plugins\solutions\Exercise 2 -- Databeans\.

Goals of this exercise

At the end of this exercise, you should be able to:

1.Retrieve data from the iSeries using an TOOLBOX FOR JAVA class.
2.Set the data on the GUI panel.
3.Test the GUI to verify the data.

Part 1: Retrieve data from iSeries

1. Open the SystemStatusEngine.java file using the TextPad editor. You can use your favorite Java editor to edit the Java files, but we'll use Windows TextPad application for this lab.

2. Add two new import statements.

   ```
   import com.ibm.as400.access.*;
   import com.ibm.as400.opnav.Monitor;
   ```

   We will use the com.ibm.as400.access classes in the TOOLBOX FOR JAVA to access the iSeries data. In addition, the com.ibm.as400.opnav.Monitor class allows you to log error messages. This is especially useful when you plug the GUI into Operations Navigator.

3. Add an instance variable to the class so that the databean knows about the iSeries that we will use to retrieve the data. You should add a private instance variable named m_sAS400 of type AS400.

   ```
   private AS400 m_sAS400;
   ```

4. Create a constructor method for the iSeries system.

   ```
   public SystemStatusEngine (AS400 anAS400) {
   m_sAS400=anAS400;
   }
   ```

5. Modify the load() method to retrieve data using the TOOLBOX FOR JAVA SystemStatus class.

   a. Add a try/catch block to contain the existing member variables (attributes). You can use the *logThrowable* method to log any errors.

   ```
   try {
     m_sCpuUtilization = “”;
     m_sSystemASPSize = “”;
     m_sSystemASPPercent = “”;
     m_sPermanentAddressesUsed = “”;
     m_sTemporaryAddressesUsed = “”;
   } catch (Exception e) {
     Monitor.logThrowable(e);
   }
   ```

   b. Modify the try/catch block. Create a new object called aStatus of type SystemStatus at the beginning of the try block.

   ```
   SystemStatus aStatus = new SystemStatus (m_sAS400);
   ```

   This new object uses the methods in the TOOLBOX FOR JAVA class called SystemStatus. To find out more information about these methods, check out the Javadoc.

c. Modify the existing assignments to the member variables within the try/catch block. Update the assignments to extract the appropriate attributes from the newly created aStatus object. For example:

```
m_sSystemASPPercent = “”;
```

would be changed to:

```
m_sSystemASPPercent = new Float
(aStatus.getPercentSystemASPUsed()).toString() + “ %”;
```

Member variable	**SystemStatus method**	**Data type**
m_sCpuUtilization	getPercentProcessingUnitUsed	Float
m_sSystemASPSize	getSystemASP	Integer
m_sSystemASPPercent	getPercentSystemASPUsed	Float
m_sPermanentAddressesUsed	getPercentPermanentAddresses	Float
m_sTemporaryAddressesUsed	getPercentTemporaryAddresses	Float

Note: Percentages are handled differently for different languages and countries. The technique used in the lab may have to be changed for a different language.

1. Save the changes and close Textpad.
2. Compile the databean from a MS-DOS prompt.

```
javac SystemStatusEngine.java
```

Didn’t work? Now what?
•Check the classpath. Does it include everything that was listed in the setup portion of the lab? Did you run the javasetup batch file before you compiled the file?
•Check SystemStatusEngine.java for typing errors. Remember that Java is case-sensitive! Also check for semi-colons and curly braces.

Remember to ask a lab attendant for help or consult Appendix A for the solutions at any time during this lab, if you get stuck.

Part 2: Test the Application

1. Open the TextPad editor. Save the new file as SystemStatusTester.java.
2. Add two import statements to the test application.

   ```
   import com.ibm.as400.access.*;
   import com.ibm.as400.ui.framework.java.*;
   ```

   We will use the com.ibm.as400.access classes in the TOOLBOX FOR JAVA to access the iSeries data. In addition, the com.ibm.as400.ui.framework.java classes contain the infrastructure classes to display the PDML panel that you created.
3. Create the class and method.

   ```
   class SystemStatusTester extends java.awt.Frame {
     public static void main (String[] args) {
       SystemStatusTester aSystemStatusTester = new SystemStatusTester();
   ```

4. Create a new AS400 object named theMachine (within the method *public static void main*).

   ```
   AS400 theMachine = new AS400();
   ```

5. Create a new SystemStatusEngine object named theSystemEngine. Pass the iSeries object as a parameter.

   ```
   SystemStatusEngine theSystemEngine = new SystemStatusEngine(theMachine);
   ```

6. Call the load() method of the SystemStatusEngine object to retrieve the data from the iSeries.

   ```
   theSystemEngine.load();
   ```

7. Create a DataBean array, which contains the SystemStatusEngine databean object.

   ```
   DataBean[] dbeans = {theSystemEngine};
   ```

8. Create a PanelManager object named pm.

   ```
   PanelManager pm = null;
   ```

   The PanelManager class allows you to display the GUI panel in the PDML file. To find out more information about this class, check out the Javadoc.

9. Add a try/catch block to instantiate the PanelManager object using the constructor method.

   ```
   try {
     pm = new PanelManager ("SystemStatus", "SystemResources", dbeans,
   aSystemStatusTester);
   } catch (DisplayManagerException e) {
     e.displayUserMessage(aSystemStatusTester);
   }
   ```

   The PanelManager constructor takes the following parameters (in this sequence)

a. A String containing the name of the PDML file. The PDML document must be in a directory or JAR file in the CLASSPATH. The PanelManager first looks for a serialized panel definition before attempting to parse the PDML file.
b. A String containing the name of the actual panel required.
c. The databean array to be used for the panel.
d. The owning frame.

10. Display the panel by running the setVisible method.

    Pm.setVisible(true);

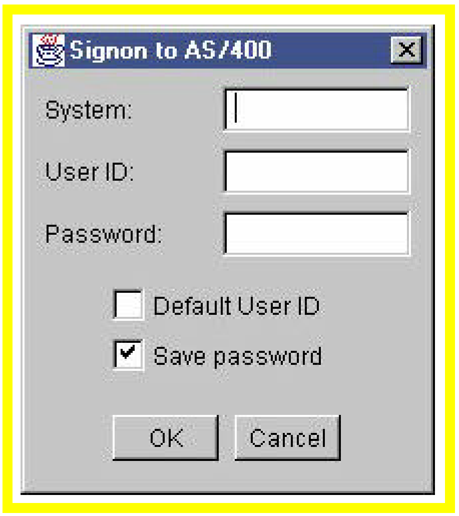
11. Save the changes and close Textpad.
12. Compile the test program from an MS-DOS prompt.

    javac SystemStatusTester.java

13. Run the test program from an MS-DOS prompt.

    java SystemStatusTester

The following logon screen will appear. Sign-on to the iSeries using the user id and password you were given for the lab.


Signon to AS/400

System:

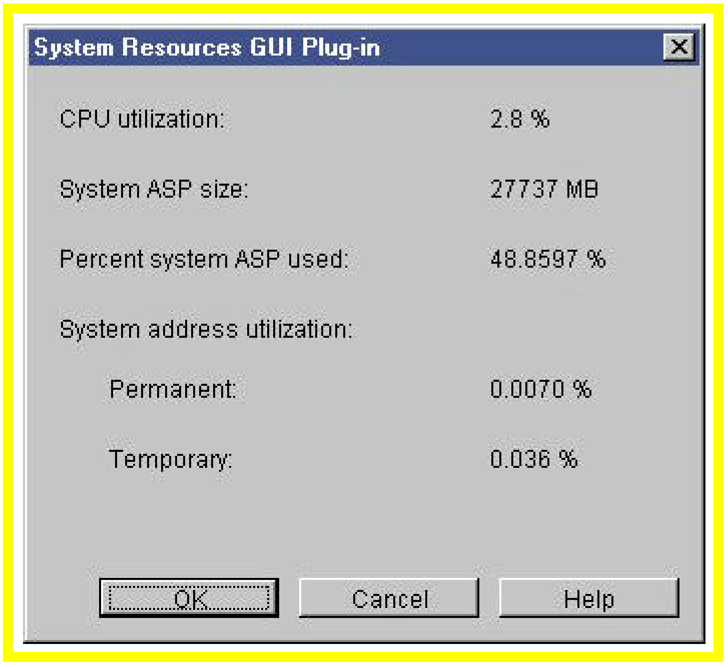
User ID:

Password:

[ ] Default User ID

[✓] Save password

After logging on to the iSeries, the SystemStatus panel will appear as follows, displaying the current data from the iSeries.



CPU utilization:	2.8 %
System ASP size:	27737 MB
Percent system ASP used:	48.8597 %
System address utilization:	
Permanent:	0.0070 %
Temporary:	0.036 %

Didn't work? Now what?

•Check the classpath. Does it include everything that was listed in the setup portion of the lab? Did you run the javasetup batch file before you compiled the file?

•Is the the PDML file is in a directory or JAR file that is in the CLASSPATH?

•Check SystemStatusTester.java for typing errors. Remember that Java is case-sensitive! Also check for semi-colons and curly braces.

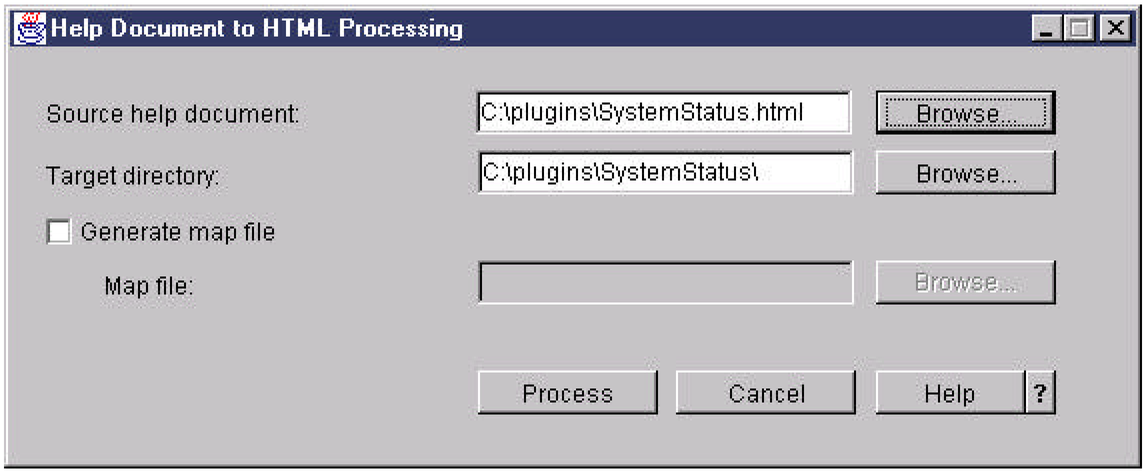
Remember to ask a lab attendant for help or consult Appendix A for the solutions at any time during this lab, if you get stuck.

Part 3: Split the Help and Test Help at Runtime

In order for help to be used at runtime, the topics within the PDML file need to be separated into individual HTML files. When you run Help Document to HTML Processing, the topics are broken into individual files and put into a subdirectory named after the Help Document and PDML file. The runtime environment expects the individual HTML files to be in a subdirectory with the same name as the Help Document and PDML file. The Help Document to HTML Processing dialog gathers the information needed and calls the HelpDocSplitter program to do the processing:

Go to a DOS prompt to run the Help Document to HTML Processing and enter:

C:\plugins\java com.ibm.as400.ui.tools.hdoc2htmViewer



Source help document: C:\plugins\SystemStatus.html

Target directory: C:\plugins\SystemStatus\

☐ Generate map file

Map file:

Now, run the test application again and test the help.

Exercise 3: Adding an Operations Navigator plugin

Introduction

In this section, you will deploy the PDML application as a plug-in to Operations Navigator. You will create the ActionsManager interface that displays context menus to a user. It is the simplest interface to write and use to extend Operations Navigator.

Once an actions manager is added into the Operations Navigator, various methods are called depending on the action of the user and the state of Operations Navigator. When implementing the ActionsManager interface, we implement these methods:

- initialize(ObjectName[] arg1, ObjectName arg2)
- queryActions(int arg1)
- actionSelected(int arg1, java.awt.Frame arg2)

These methods are covered in detail in the following steps.
The completed Java code can be found in C:\Plugins\Solutions\Exercise 3 -- ActionManager.

Goals of this excercise:

At the end of this exercise, you should be able to:

1.Create an ActionsManager interface. Within the interface you will:
 -- register the SystemStatusManager as a context manager of the iSeries Network
 -- define a queryAction method
 -- define an actionSelected method

2.Modify the windows registry

3.Register your plugin

4.Test the extension

Part 1: Create the SystemStatusManager.java

1. Open TextPad and create a new Java source file named **SystemStatusManager.java**.

2. Add three import statements.

We import all the classes in the following packages:

```
import com.ibm.as400.opnav.*;
import com.ibm.as400.ui.framework.java.*;
import com.ibm.as400.access.*;
```

3. Define a new public class definition named SystemStatusManager that extends the Object class and implements ActionsManager.

```
class SystemStatusManager extends Object implements ActionsManager {
```

4. We define two private class variables:

- initObjs — An array of ObjectName
- dragDropObj — Of type ObjectName

```
private ObjectName[] initObjs;
private ObjectName dragDropObj;
```

5. Define an initialize method.

When Operations Navigator detects that a plug-in or extension may be required to perform an action, it calls the initialize method. The initialize method accepts two variables as parameters and returns void. The first parameter is an array of ObjectName that identifies objects on which the ActionsManager implementor may want to perform some actions. The second parameter is a single ObjectName that is used if an Operations Navigator object is dragged or dropped onto another Operations Navigator object. In this case, we register the SystemStatusManager as a context manager of the iSeries Network. This way, it is called when the end user right clicks on an iSeries system. As such, the only type of Operations Navigator ObjectName we should receive is an "AS4" type.

The parameters that are passed should be stored in the class variables. A completed definition of this method appears as shown here:

```
public void initialize(com.ibm.as400.opnav.ObjectName[] arg1,
com.ibm.as400.opnav.ObjectName arg2) {
initObjs = arg1;
dragDropObj = arg2;
}
```

6. Define a queryAction method.

The queryAction method is used by Operations Navigator to ask the ActionsManager to report on the actions it can perform and what is displayed to the user. To inform Operations Navigator, the queryActions method returns an array of ActionDescriptor objects. An ActionDescriptor object has a number of attributes, some of which are shown in the following

Attribute	Methods	Comment
Help Text	set, get	Action help test displayed I the status field in the Operations Navigator main window
Text	set, get	Text displayed in the menu
Verb	set, get	A non-displayed stirng used to identify the action.
ID	get, set & ActionDescription(ID)	A numeric integer used to identify the action. All acitons should be assigned a unique ID for any ActionManager implementor.
subActions	set, get	Enables context menus within menu items
Default	set, isDefault()	If true, then this menu item is the default action on this context menu.

In this case, the queryActions method needs to return an array of one element, since we are only performing one action. The ActionDescription returned needs to have the attributes set that are shown in the following table.

Property	Required Value
ID	1
Text	System Status
Help Text	Loads the lab System Status plugin
Verb	ITSOSysSts

Before setting up the ActionDescription, ensure that you are defining the correct menu to display in the given context. This is achieved by checking the ObjectType that was set when the initialize method is called. In this case, we registered the context menu to exist only under an iSeries System. This object type is AS4, and it represents an iSeries network connection. For a complete list of the available types, see the system registry entries for

HKEY_CLASSES_ROOT\IBM.AS400.Network\TYPES.

The getObjectType method can throw an ObjectNameException so we encapsulate the code within a **try** block. The following code sample illustrates all of these points:

```
public ActionDescriptor[] queryActions(int arg1) {
  ActionDescriptor[] actions = new ActionDescriptor[0];
  String objType = null;
  try {
    objType = initObjs[0].getObjectType();
    if (objType.equals("AS4")) {
    if ((arg1 & CUSTOM_ACTIONS) == CUSTOM_ACTIONS) {
    actions = new ActionDescriptor[1];
    ActionDescriptor act = new ActionDescriptor(1);
    act.setText("System Status");
    act.setHelpText("Loads the LAB System Status Plugin");
    act.setVerb("ITSOSysSts");
    actions[0] = act;
    }
   }
  }
 catch (Exception e) {
  Monitor.logThrowable(e);
  }
 return actions;
}
```

7. Define an actionSelected method.

The actionSelected method is only called by Operations Navigator when the user selects one of the actions returned in queryActions. For this reason, we need to instantiate any databeans and code the majority of the plug-in within this method:

a. We add a public method called actionSelected that accepts two arguments. The first argument should be an integer called arg1. The second should be a Frame called arg2. The method does not return a value.

b. We use an **if** clause to verify that the action selected (arg1) is equal to 1. Since this is the only value set by queryActions, it should always be set to 1.

c. Once the action is verified, we use a **try** block since the subsequent methods may throw an exception.

d. Within the **try** block we extract the iSeries system object from the first element in the initObjs variable using the getSystemObject(). We store this value in a new AS400 object called theMachine.

e. We use theMachine to create a new instance of the SystemStatusEngine databean and call it theStatusEngine.

f. We have the theStatusEngine load data by calling the load() method.

g. We create an array called dBeans of type DataBean. We set the first and only element to be theSystemStatusEngine. Now that the databean is initialized, we need to display the SystemResource panel. To achieve this, we use the PanelManager class and construct a panel manager object using the databean and the owning frame.

h. We declare a PanelManager object called pm and set it to null.

i. Within a **try** block, we instantiate the pm object using the constructor method, passing the following parameters in sequence:
   - A String containing the name of the PDML file. The PDML document must be in a directory or JAR file in the CLASSPATH. The PanelManager will first look for a serialized panel definition before attempting to parse the PDML file.
   - A String containing the name of the actual panel required.
   - The array of databeans to be used by this panel.
   - The owning frame.

j. If the PanelManager constructor method throws a anelManagerException**,** we catch it and use the displayUserMessage() method, to display a message to the user.

k. To display the new panel and hand over control to this panel to the user, we use the setVisible(true) method.

l. We use a catch to trap all exceptions and log them to the Monitor log using the logThrowable() method.

After completing this step the defined method should appear like this example:

```
public void actionSelected(int arg1, java.awt.Frame arg2) {
    if (arg1 == 1)
    {
        try {
            AS400 theMachine = (AS400)initObjs[0].getSystemObject();
            SystemStatusEngine theSystemEngine = new
            SystemStatusEngine(theMachine);
            theSystemEngine.load();
            DataBean[] dbeans = {theSystemEngine};
            PanelManager pm = null;
            try
            {
            pm = new
            PanelManager("SystemStatus","SystemResources",dbeans,arg2);
            }
            catch (DisplayManagerException e){
            e.displayUserMessage(arg2);
            }
            pm.setVisible(true);
        }
        catch (Exception e) {
            Monitor.logThrowable(e);
        }
     }
    }
```

8. Save the java source.

9. Compile the java source. javac SystemStatusManager.java

See C:\Plugins\solutions\Exercise 3 -- ActionManager for the solution code

Didn't work? Now what?

•Check the classpath. Does it include everything that was listed in the setup portion of the lab? Did you run the javasetup batch file before you compiled the file?

•Check SystemStatusManager.java for typing errors. Remember that Java is case-sensitive! Also check for semi-colons and curly braces.

Remember to ask a lab attendant for help or consult Appendix A for the solutions at any time during this lab, if you get stuck.

Part 2: Modifying the Windows Registry

To reduce the possibility of incorrectly setting the registry, use a registry file. An incorrectly set registry can, under extreme circumstances, causes your machine to stop functioning. Therefore, we recommend that you always backup the registry prior to any modification. Follow these steps:

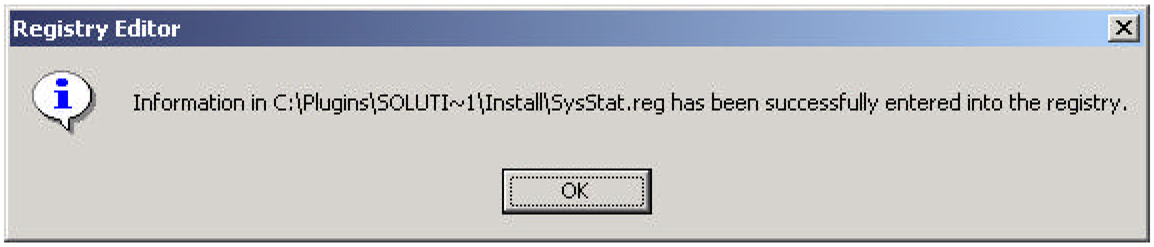
1. Backup the registry.
   a. Start the regedit program
   b. Select Registry --> Export Registry File from the menu and export all entries to a file called Original in the C:\plugin directory.

2. The sample registry file (**SysStat.reg**) exists in C:\Plugins\Solutions\Install. The following is the code for the registry file.

```
 REGEDIT4
;
; Define the primary registry key for the plugin
;
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\LAB.SystemStatusPlugin]
;
; You have written an extension to the shell, so use EXT for the type of
; plugin. If a ListManager interface had been implemented then you would
; have set this to Plugin
;
"Type"="EXT"
"MinimumRISCRelease"="ANY"
"MinimumIMPIRelease"="NONE"
"ProductID"="NONE"
"ServerEntryPoint"="NONE"
; The next value point to the base directory or archive used to find any
; required user class definitions.
"JavaPath"="C:\\plugins\\solutions\\Exercise 3 -- ActionManager;"
"JavaMRI"=""
;
; Although no DLL is used you are required to add a registry entry. It will
; never be used or accessed but it MUST exist.
;
"NLS"="SysStat.dll"
"NameID"=dword:00000000
"DescriptionID"=dword:00000000
;-------------------------------------------------------------------
;Register a context menu handler for the new folder and its objects
;
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\LAB.SystemStatusPlugin\shellex\AS/400
Network\AS/400 System\ContextMenuHandlers\{1827A857-9C20-11d1-96C3-00062912C9B2}]
"JavaClass"="SystemStatusManager"
```

3. Go to this directory in the Windows NT Explorer, and double click on the **SysStat.reg** file to have the system apply it to the Windows registry.

   You should receive the following message:

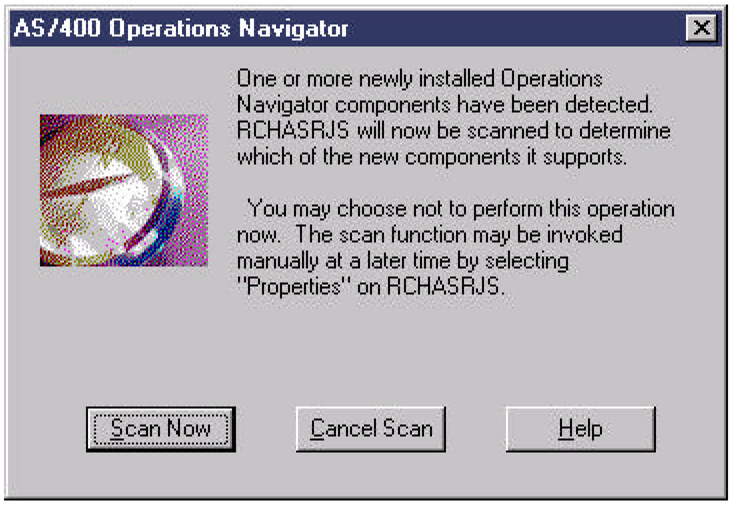

Registry Editor

Information in C:\Plugins\SOLUTI~1\Install\SysStat.reg has been successfully entered into the registry.

Part 3: Test the Extension

To test the extension, perform the following steps:

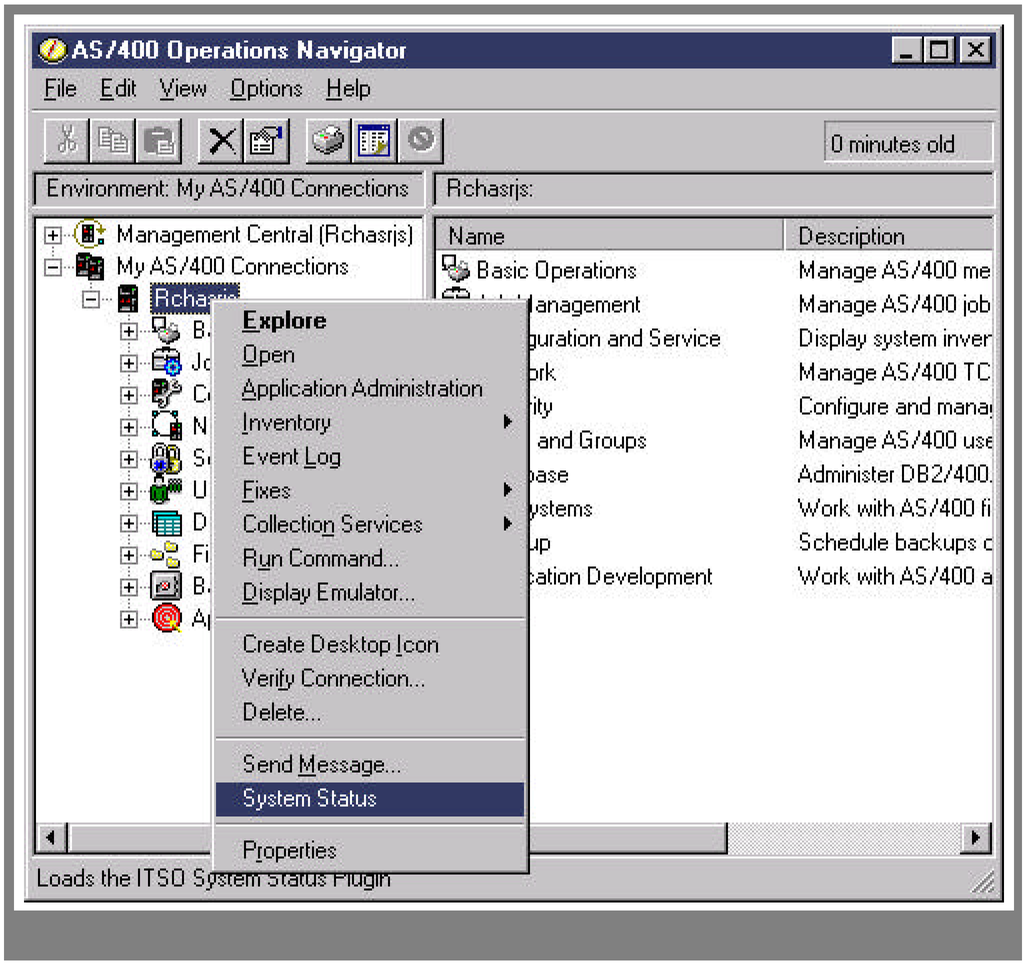
1. Start Operations Navigator. If it is already active, then close and re-open it to force it to read the registry again.
2. Select and expand the defined iSeries system.
3. Log on to the system as prompted.
4. If the extension is registered, a scan operation takes place. Click the Scan Now button to confirm that the scan can occur now.


AS/400 Operations Navigator

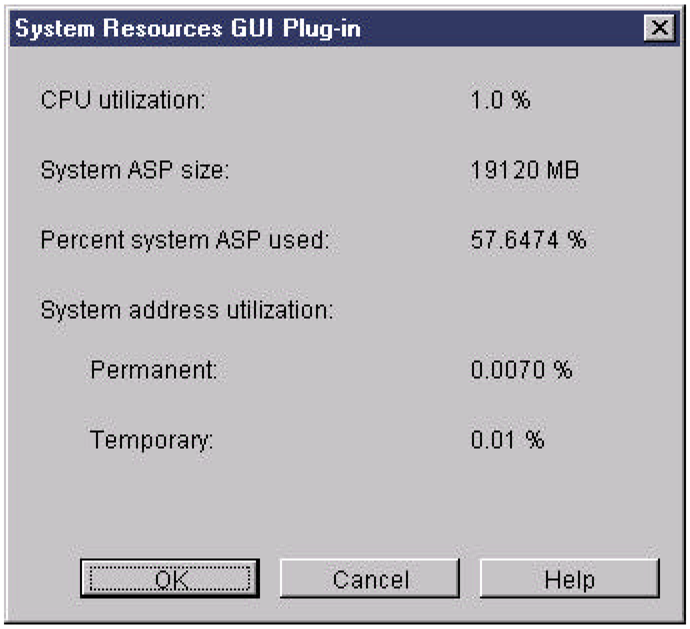
One or more newly installed Operations Navigator components have been detected. RCHASRJS will now be scanned to determine which of the new components it supports.

You may choose not to perform this operation now. The scan function may be invoked manually at a later time by selecting "Properties" on RCHASRJS.

5. Right click on the system menu item to see that the newly installed extension is available in the System's context menu.



6. Select the menu option System Status. After retrieving the data from the iSeries system, the System Resource panel is displayed.

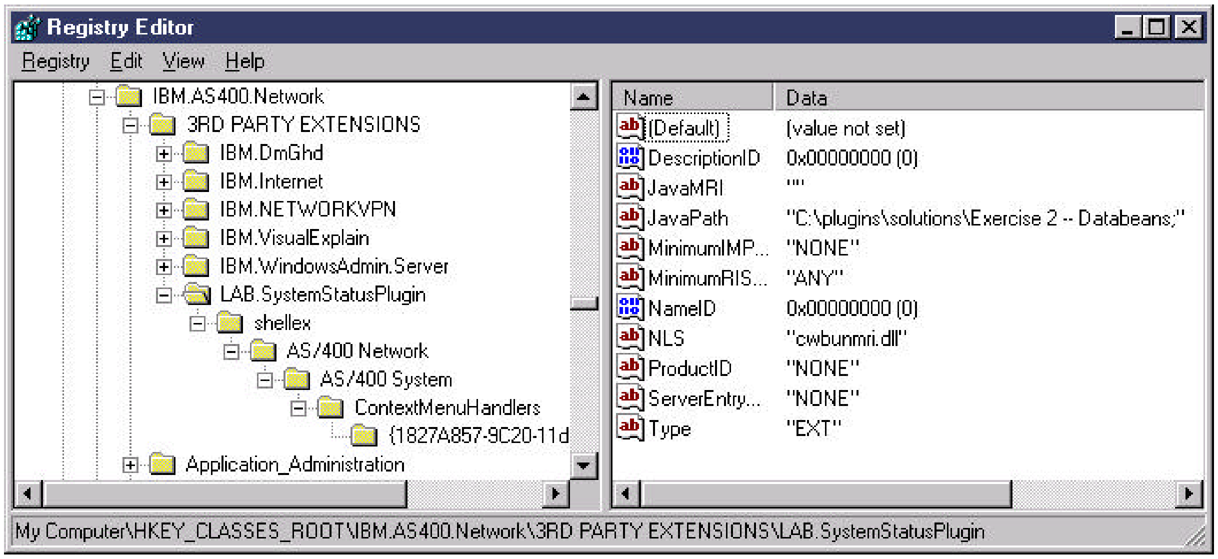


CPU utilization:	1.0 %
System ASP size:	19120 MB
Percent system ASP used:	57.6474 %
System address utilization:	
Permanent:	0.0070 %
Temporary:	0.01 %

NOTE: Be extremely cautious with the Windows Registry, you can cause catastrophic damage!

To verify in the registry:

1. Start
2. Run
3. Regedit
4. HKEY_CLASSES_ROOT
5. Edit, find, search for ‘3rd’
6. Notice our LAB.SystemStatusPlugin in the registry



Name	Data
(Default)	(value not set)
DescriptionID	0x00000000 (0)
JavaMRI	""
JavaPath	"C:\plugins\solutions\Exercise 2 -- Databeans;"
MinimumIMP...	"NONE"
MinimumRIS...	"ANY"
NameID	0x00000000 (0)
NLS	"cwbunmri.dll"
ProductID	"NONE"
ServerEntry...	"NONE"
Type	"EXT"

My Computer\HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\LAB.SystemStatusPlugin

Exercise 4: Installing your Plug-in

Introduction

In this exercise, you will learn how to deliver your Java plug-in code to Operations Navigator. You will learn where the application files are stored on the iSeries, how Client Access Selective Setup detects your application for installation, and how your plugin is registered, or made known to Operations Navigator.

You can deliver your Java plug-in code to Operations Navigator users by including it with your OS/400 applications. The install program for the application writes the plug-in's code binaries, registry file, and translatable resources to the **/QIBM/UserData/OpNavPlugins** folder in the iSeries Integrated File System (iSeries IFS). Once this process is completed, users can obtain the plug-in from the iSeries IFS (with the help of an iSeries NetServer mapped network drive) by invoking the Client Access Selective Setup program. The setup program 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. All you need is a setup file, which identifies the files to be installed. If you provide a Windows policy template with your plug-in, you also can take advantage of Windows system policies to control which network users can install your plug-in. You also can use the iSeries-based Application Administration support of Operations Navigator
to control which users and groups have access to your plug-in.

Additional resources for Application Administration support information:

- Application Administration support for your plug-in
- Application Administration topic in the Information Center

After the users have installed your new plug-in, you may choose either to upgrade it at a later date, or to ship code problem fixes. When the code is upgraded on the iSeries, the Client Access Check Version program detects that this process has occurred, and automatically downloads the updates to the users' PCs. Client Access also provides uninstall support, which allows users to
completely remove the plug-in from their PCs.

Users can display the plug-ins that are installed on their PCs by selecting the Plug-ins tab on the Operations Navigator Properties for a particular iSeries.

Goals of this exercise

1. Place your application files in the correct location in the IFS file system on the iSeries
2. Use Client Access Selective Install to detect your plugin and install it.
3. Determine what Plugins are installed on the iSeries

NOTE: This exercise assumes that NetServer is setup. To setup NetServer

1. Click Start > Find >Computer
2. Enter the iSeries NetServer name to find iSeries NetServer in the network
3. Once found, double-click the iSeries NetServer icon to list the curretly shared resources
4. Right-click on the shared resouce you would like to map a drive to and selct Map Network Drive.
5. Select the drive letter you would like and press enter

The following files are required to make your plugin known to Operations Navigator. These files are provided in C:\Plugins\Solutions\Install

1. **Setup.ini -** File that provides the information needed to install, upgrade, and service an Operations Navigator plug-in.
2. **mri.dll -** Provides the icon and text that is displayed for your plug-in (i.e. the icon and description in the Op Nav tree) **The file provided is SysStat.dll**
3. **filename.reg -** Contains the registry entries needed for an Operations Navigator plug-in written in Java. The registry entries inform the Navigator that a Java plug-in exists, and identify the names of the Java classes which implement the interfaces defined for a plug-in. **The file provided is SysStat.reg**

Part 1: Copy files to iSeries

1. On the iSeries in Operations Navigator, go to **File Systems, Integrated File System, Root**
2. Then drill down to the following directory: **QIBM/UserData/GUIPlugins**
3. Create directory **PluginLab.SystemStatus**
4. In Windows Explorer, go to **C:\Plugin\solution\Exercise 3 -- ActionManager** and copy the following files to **QIBM/UserData/GUIPlugins /PluginLab.SystemStatus**
   a. **setup.ini** and
   b. **SysStat.reg** files to
   c. **SystemStatusEngine.class**
   d. **SystemStatusTester.class**
   e. **SystemStatusManager.class**

NOTE: we are doing individual class files here, but you could create a .jar file and place the .jar file in QIBM/UserData/GUIPlugins/PluginLab.SystemStatus along with the setup.ini and SysStat.reg files

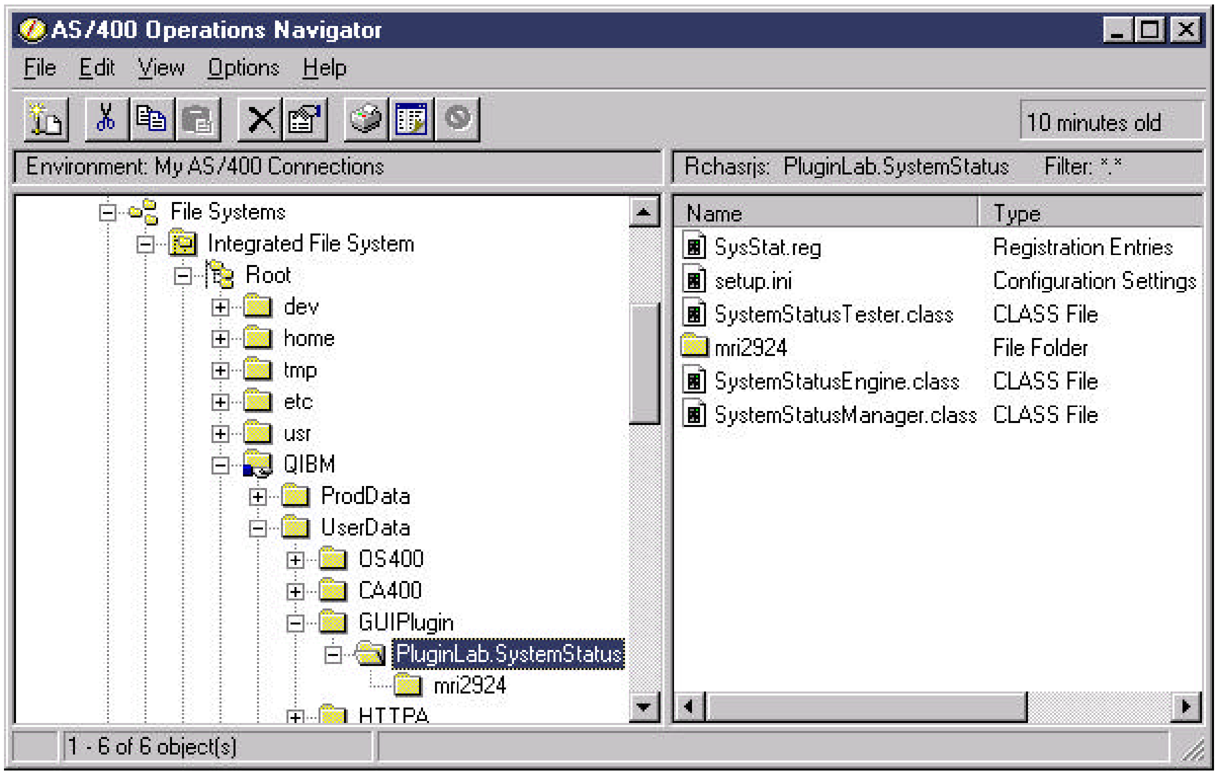
5. On the iSeries, in **QIBM/UserData/GUIPlugins/PluginLab.SystemStatus** create directory **mri2924;**

Note: an mri29xx directory must be created for each national language you support.

6. Change directories to **QIBM/UserData/GUIPlugins/PluginLab.SystemStatus/mri2924**
7. Copy the **SysStat.dll** and **mrisetup.ini** files from

   C:\Plugin\solution\Exercise 3 -- ActionManager to the **QIBM/UserData/GUIPlugins/PluginLab.SystemStatus/mri2924** directory.

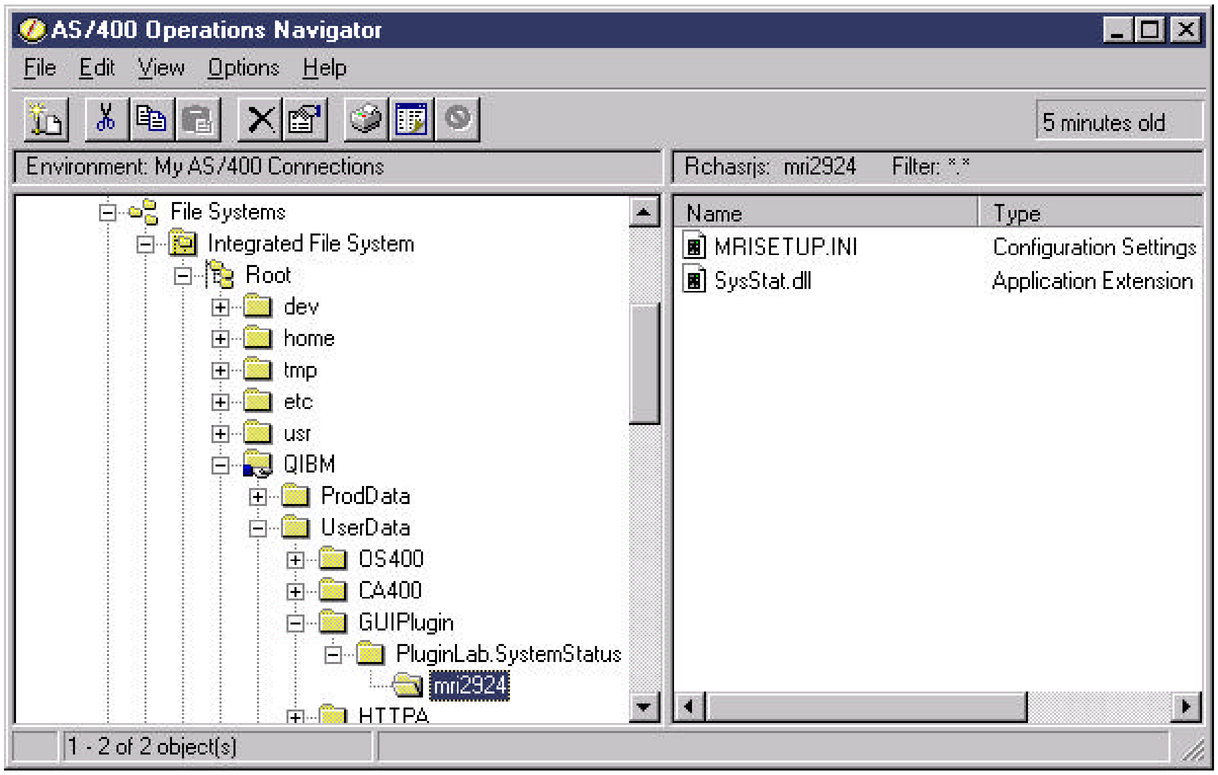
The following figure shows the files in the IFS directory structure:



Name	Type
SysStat.reg	Registration Entries
setup.ini	Configuration Settings
SystemStatusTester.class	CLASS File
mri2924	File Folder
SystemStatusEngine.class	CLASS File
SystemStatusManager.class	CLASS File

NOTE: Directory QIBM/UserData/**GUIPlugins** is used for applications that run on both the XD1 and Express clients. Directory QIBM/UserData/**OpNavPlugins** is used for applications that only run on the express client.

The following screen shows the MRISETUP.ini and SysStat.dll in the mri2924 directory:



Name	Type
MRISETUP.INI	Configuration Settings
SysStat.dll	Application Extension

Part 2: Clean Up your PC registry and directories

Since you have already registered your the SystemStatusManager in Excercise 4, you will have to clean up the registry entries and the Client Access directory: Remember to be extremely careful modifying anything in the windows registry.

1. Clean up the registry;
   - Regedit
   - HKEY_CLASSES_ROOT
   - Edit, find, **LAB.SystemStatusPlugin** (delete all instances; should be 3)

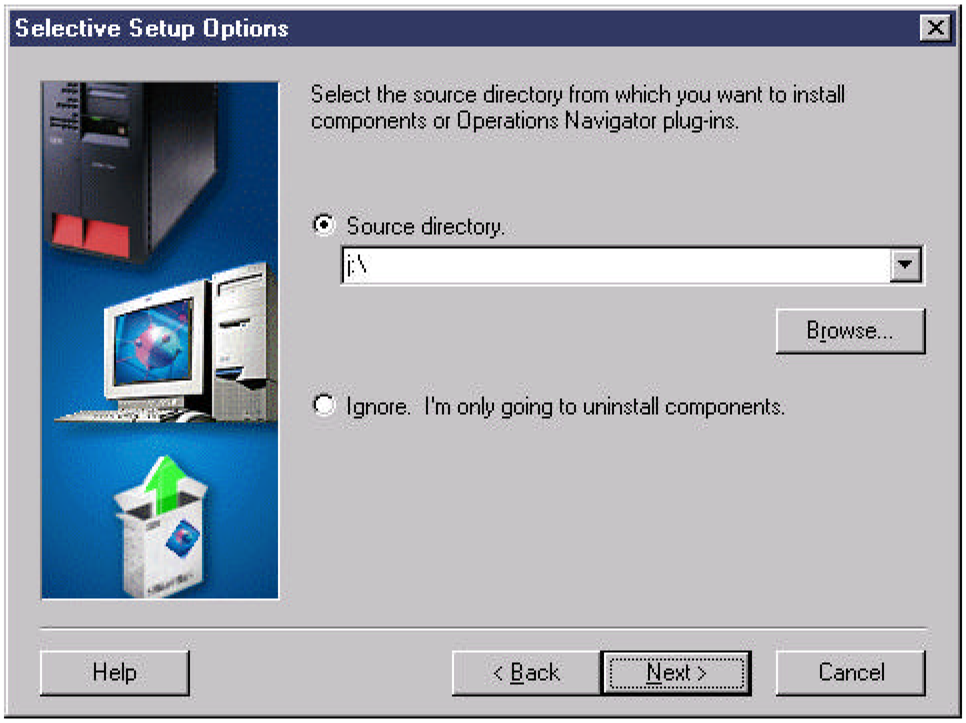
2. Clean up the Client Access directory (NOTE: Don't do this until the end)
   - C:\Program Files\IBM\Client Access\Plugins - delete the **PluginLab.SystemStatus** directory and all it's contents
   - C:\Program Files\IBM\Client Access\Mri2924 - delete the **PluginLab.SystemStatus** directory and all it's contents

Part 3: Run Client Access Selective Setup to install the Application

The next step is to perform Client Access Selective Setup to install the application that is now on the iSeries to the Client.

1. Go to the Client Access folder
2. Select 'Selective Setup'
3. Click on 'Next' on the add components screen
4. Make sure that you are mapped to your NetServer drive for your source directory (in the case of the following example, the j:\ driver is mapped to the netserver)

The source directory should be: **NetServerSystemName/QIBM/GUIPlugin**



Selective Setup Options

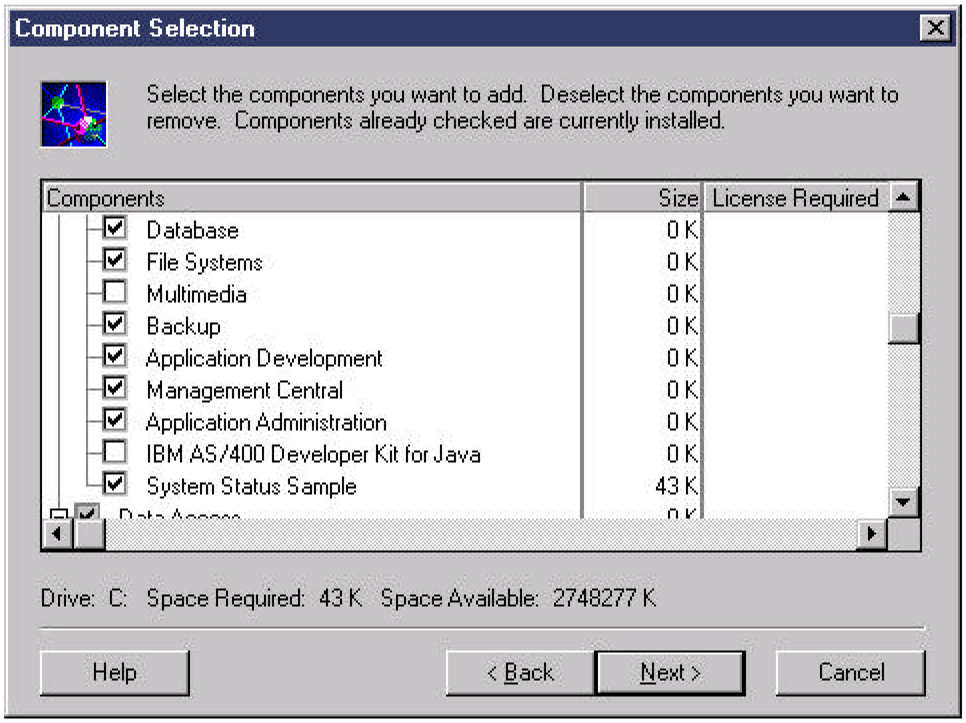
Select the source directory from which you want to install components or Operations Navigator plug-ins.

[✓ Source directory.] j:\

Ignore. I'm only going to uninstall components.

5. Continue on through selective setup

6. On the component selection screen, is your System Status Sample application in the list?



Select the components you want to add. Deselect the components you want to remove. Components already checked are currently installed.

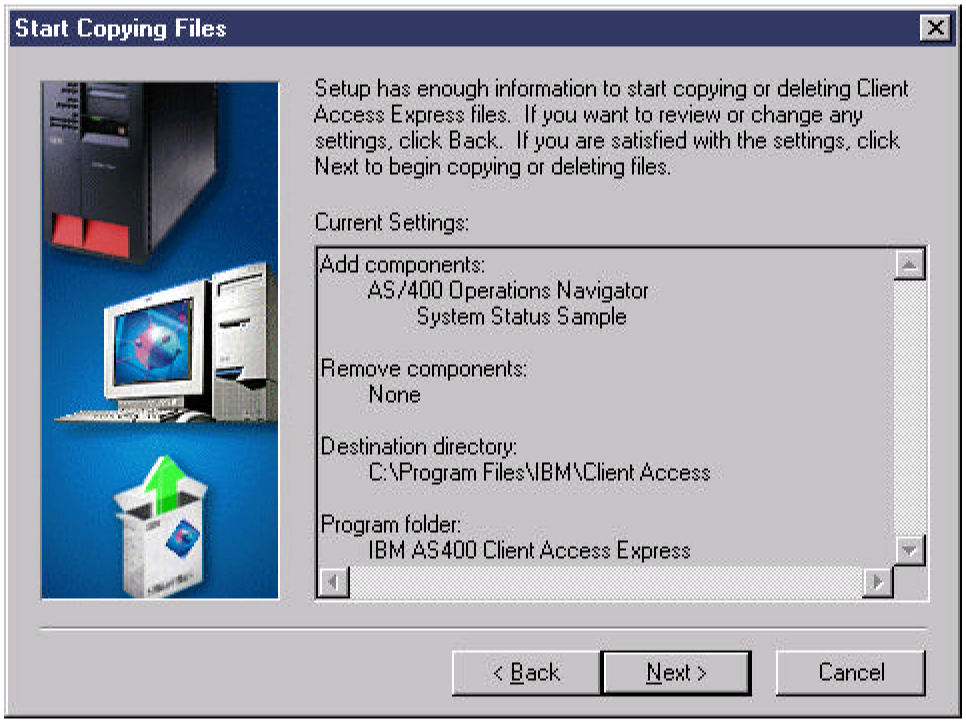
Components	Size	License Required
✓ Database	0 K	
✓ File Systems	0 K	
Multimedia	0 K	
✓ Backup	0 K	
✓ Application Development	0 K	
✓ Management Central	0 K	
✓ Application Administration	0 K	
IBM AS/400 Developer Kit for Java	0 K	
✓ System Status Sample	43 K	

Drive: C: Space Required: 43 K Space Available: 2748277 K

7. Select (check) the System Status Sample

Note: if the checkbox next to your System Status sample sample is already checked, that means it was already registered. You will have to make sure that all instances of **LAB.SystemStatusPlugin** are deleted from both the Windows Registry and all instances of **PluginLab.SystemStatus** are deleted from the Client Access directory. See previous cleanup instructions.

Select 'Next' and continue on through seclective setup



Setup has enough information to start copying or deleting Client Access Express files. If you want to review or change any settings, click Back. If you are satisfied with the settings, click Next to begin copying or deleting files.

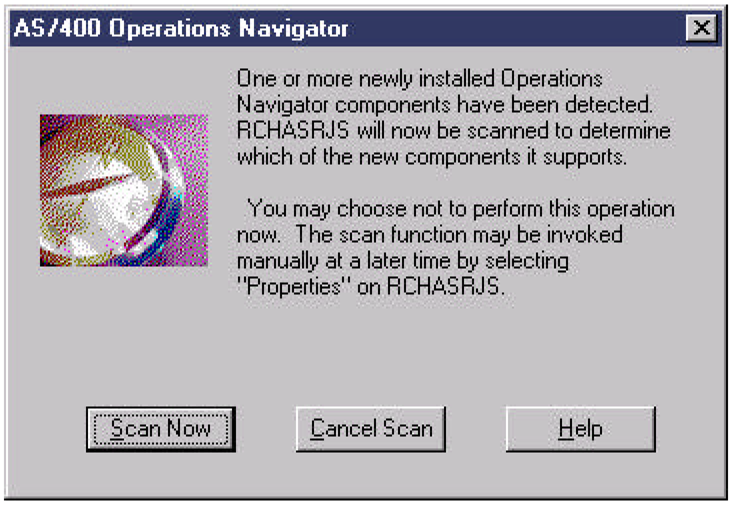
Current Settings:

```
Add components:
      AS/400 Operations Navigator
            System Status Sample

Remove components:
      None

Destination directory:
      C:\Program Files\IBM\Client Access

Program folder:
      IBM AS400 Client Access Express
```

8. On the install complete screen, deselect to view the readme and add a shortcut.
9. Select Next
10. Select OK
11. SelectFinish

Part 4: Test the installation

1. Start Operations Navigator
2. Select the system and sign on
3. The scan screen should be displayed; Select scan now



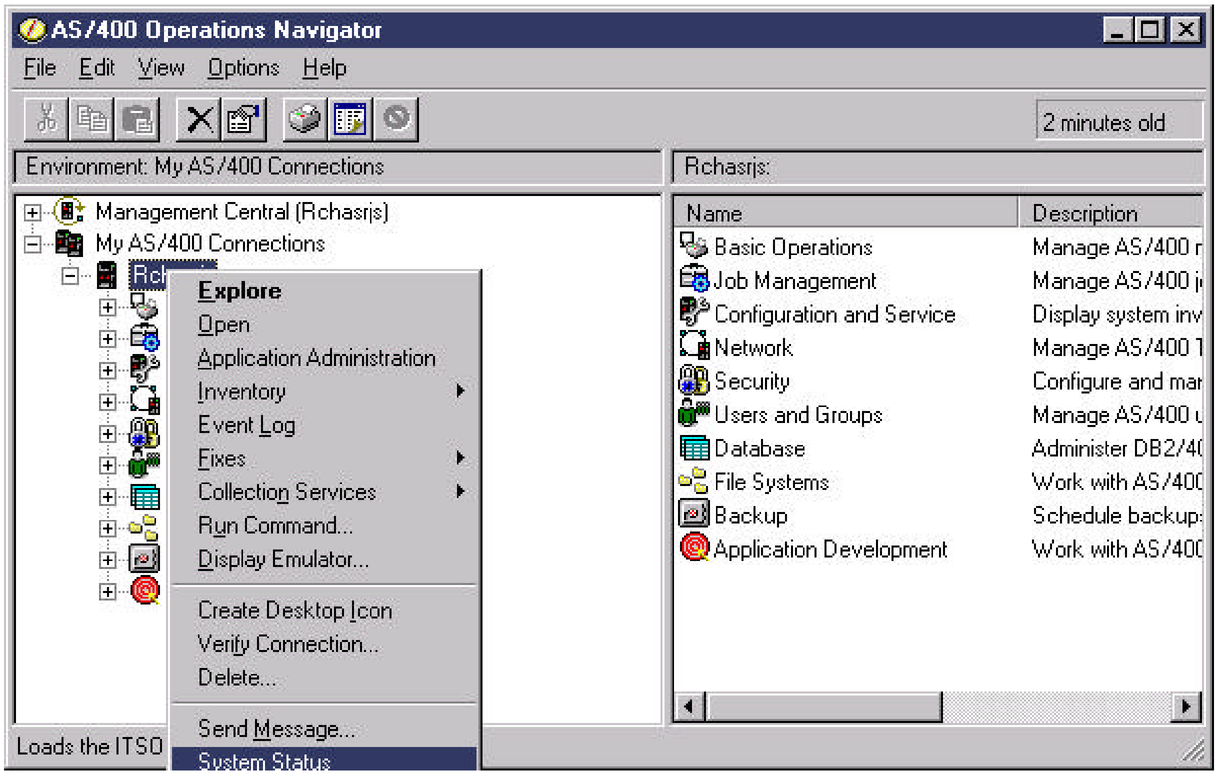
AS/400 Operations Navigator

One or more newly installed Operations Navigator components have been detected. RCHASRJS will now be scanned to determine which of the new components it supports.

You may choose not to perform this operation now. The scan function may be invoked manually at a later time by selecting "Properties" on RCHASRJS.

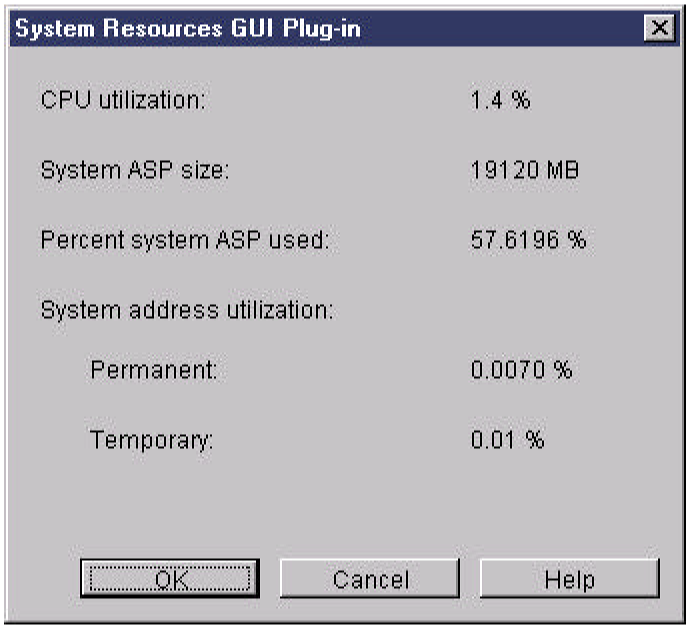
4. Right click on the system name.

Is your System Status application listed in the context menu?



Name	Description
Basic Operations	Manage AS/400 r
Job Management	Manage AS/400 j
Configuration and Service	Display system inv
Network	Manage AS/400 T
Security	Configure and mar
Users and Groups	Manage AS/400 u
Database	Administer DB2/40
File Systems	Work with AS/400
Backup	Schedule backup:
Application Development	Work with AS/400

5. Select System Status



CPU utilization:	1.4 %
System ASP size:	19120 MB
Percent system ASP used:	57.6196 %
System address utilization:	
Permanent:	0.0070 %
Temporary:	0.01 %

Didn’t work? Now what?

- Did you Clean up the Windows registry and the Client Access directories prior to registering your plugin?
- Is your SystStat.reg file correct?
- Is your Setup.ini file correct?
- Do you have all the necessary files installed in the correct directories on the iSeries?

Exercise 5: Retrieving and Setting iSeries Data using PCML

Introduction

In Exercise 2, you retrieved iSeries data using the iSeries Toolbox for Java's access classes. In this exercise, you will retrieve the same data, but this time you will retrieve it using the Program Call Markup Language (PCML) support (also in the TOOLBOX FOR JAVA).

The TOOLBOX FOR JAVA provides many classes to retrieve common iSeries objects (for example, User, Job, and DataArea). For these objects, Toolbox presents an easy-to-use Java interface. Under the covers, Toolbox implements these interfaces (ProgramCall) by calling iSeries APIs. When writing these objects, the Toolbox developers realized ProgramCall is too hard to use to call complex APIs. PCML, an implementation of XML, was created to solve the complexity problem. PCML offers the following advantages over ProgramCall:

•Parameter formats are described via html-like syntax. You no longer have to count offsets. You describe the format of the data and the PCML does the offset calculation for you. This is especially useful for variable length and nested structures. PCML does the messy math for you.

•PCML does data conversion for you. You describe the type and size of your data and PCML does the conversion for you.

•Parameter formats can be changed without re-compiling the application. The PCML source file is parsed at run-time. If you need to change it, you simple change the .pcml file and rerun the application.

Goals of this exercise

At the end of this exercise, you should be able to:

1. Find the iSeries API specifications.
2. Create PCML for the API.
3. Hook the PCML into the Java application and display the data in the PDML panel.
4. Test the PCML by running the Java application

Part 1: Create the PCML file

1. Find the specification for the API that retrieves the system status from the iSeries. (To save time, we've copied the specification to the lab PCs: C:\plugins\qwcrssts.htm)

   > The iSeries Information Center contains specifications for the OS/400 APIs. You can access the iSeries Information Center at http://www.ibm.com/eserver/iseries/infocenter (to access the API specifications, click on Programming --> CL and APIs --> OS/400 APIs). The API to retrieve system status is located within the Work Management category.

2. Open the TextPad editor. Save the new file as qwcrssts.pcml.

3. Add two import statements to the pcml file.

   import com.ibm.as400.access.*;
   import com.ibm.as400.data.*;
   We will use the com.ibm.as400.access classes in the TOOLBOX FOR JAVA to access the iSeries. The com.ibm.as400.data package contains the PCML classes.

4. Add the <PCML> start and end tags. All other tags will be inserted between these two tags.
   <pcml version=”1.0”>

   </pcml>

5. Create the structure that specifies the data format for the data we want to retrieve. The format name that contains the information we need is SSTS0200.
   <struct name=”SSTS0200”>

   </struct>

6. Within the structure, insert the data tags. These data tags identify the data that you want to retrieve and the location (offset) of that data.

   a. Insert a data tag that reserves space for the data that is returned by the API, but that we will not use in the GUI. (we will not associate a name with this tag since we aren't using the data that is returned.)
      <data type="byte" length="32" />

      > This tag reserves 32 bytes of memory for the data that we will not use. The first set of data that we want to use starts on offset 33 (on the 33rd byte).

   b. Insert a data tag that retrieves the CPU Utilization data.
      <data name=”CpuUtilization” type="int" length="4" />

Tip: Name the data in the PCML the same name as the databean attribute in the PDML. It is easier to reference the data later.

Tip: In the API specification, whenever the data type is BINARY, the data type = "int" in the PCML.

c. Insert the rest of the data tags.

Data name	**Type**	**Length**
(reserved space)	byte	4
PermanentAddressesUsed	int	4
TemporaryAddressesUsed	int	4
SystemASPSize	int	4
SystemASPPercent	int	4
(reserved space)	byte	12

7. Create the program that specifies the parameters for the API.

```
<program name="qwcrssts_SSTS0200" path="/QSYS.LIB/QWCRSSTS.PGM">

</program>
```

Tip: All OS/400 APIs are stored in QSYS.LIB.

8. Within the program tags, insert the data tags that specify the parameters and initial values for the API.
   a. Insert the data tag for the *receiver variable* parameter of the API. This tag references the structure that we just created as the receiver of the data from the iSeries.
   ```
   <data name="receiver"  type="struct" usage="output" struct="SSTS0200"/>
   ```
   b. Insert the data tag for the *length of receiver variable* parameter of the API. This tag specifies the length of all of the data (in bytes) that will be received from the iSeries (specified in the structure).
   ```
   <data name="receiverLength" type="int" length="4" usage="input" init="68"/>
   ```
   c. Insert the data tag for the *format name* parameter of the API. This tag references the structure that we just created as the receiver of the data from the iSeries.
   ```
   <data name="format"  type="char" length="8"  usage="input"  init="SSTS0200"/>
   ```
   d. Insert the data tag for the *reset status statistics* parameter of the API.

<data name="resetStatus" type="char" length="10" usage="input" init="*NO" />

e. Insert the data tag for the *error code* parameter of the API.
<data name="errorCode" type="int" length="4" usage="input" init="0"/>

9. Save the file and close Textpad.

Remember to ask a lab attendant for help or consult Appendix A for the solutions at any time during this lab, if you get stuck.

The PCML file should contain one structure and one program for each format in the API. If you wanted to retrieve more status information, you could add a structure for each of the two other formats. You would also need to add a program for each structure. As you've probably noticed, this lab has used the program name *APIName_FormatName* for easier management.

Part 2: Test the PCML (outside of the GUI)

1. Open the TextPad editor. Save the new file as qwcrssts.java.
2. Add two import statements to the test application.

```
import com.ibm.as400.access.*;
import com.ibm.as400.data.*;
```

   We will use the com.ibm.as400.access classes in the TOOLBOX FOR JAVA to logon to the iSeries. In addition, the com.ibm.as400.data package contains the classes to run the PCML file that you created.

3. Create the class and method.

```
public class qwcrssts {
  public static void main (String[] argv) {
  }
}
```

4. Create a new object named value that will receive the iSeries data.

```
    Object value;
```

5. Create a new AS400 object named theMachine (within the method *public static void main*).

```
    AS400 theMachine = new AS400();
```

6. Add a try/catch block to retrieve the PCML data and display the data.

```
try {
     ProgramCallDocument pcmlDoc = new ProgramCallDocument(theMachine, “qwcrssts”);
     PcmlDoc.callProgram(“qwcrssts_SSTS0200”);

      System.out.println(“Retrieving AS/400 data...”);

} catch (PcmlException e) {
      System.out.println(e.getLocalizedMessage());
      e.printStackTrace();
      System.out.println("*** Call to QWCRSSTS failed. ***");
      System.exit(0);
}
```

The ProgramCallDocument constructor takes the following parameters (in this sequence).
-- The iSeries to retrieve the data from.
-- The name of the PCML file (without the .pcml extension).

7. Within the try block, retrieve the value for CPU Utilization. You will retrieve the value using the ProgramCallDocument.getValue() method. When accessing values from the ProgramCallDocument class, you must specify the fully qualified name of the <data> tag. The qualified name is a concatenation of the names of all the containing tags with each name separated by a period (e.g. *ProgramName.ReceiverParameterName.DataName*)

```
value = ((((Integer) pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.CpuUtilization")).floatValue())/10);
System.out.println("    CPU Utilization:  " + value + " %");
```

8. Within the try block, retrieve the rest of the system status values and print them using System.out.println(). You will need to refer to the API specification to find out how the data is being returned. The percentage values need to be converted before they are displayed correctly (for example, CPU Utilization was received from the iSeries in tenths, so it needed to be divided by 10).

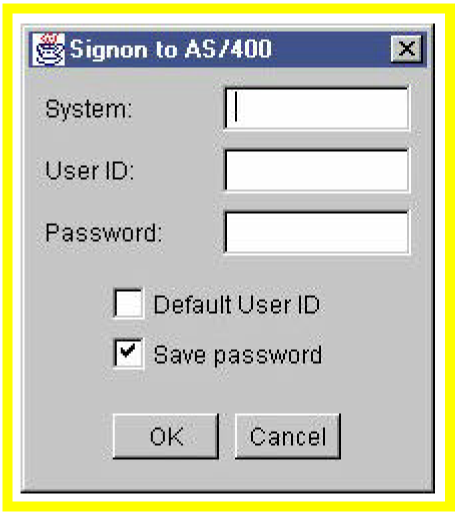
9. Compile the test program from an MS-DOS prompt.

```
javac qwcrssts.java
```

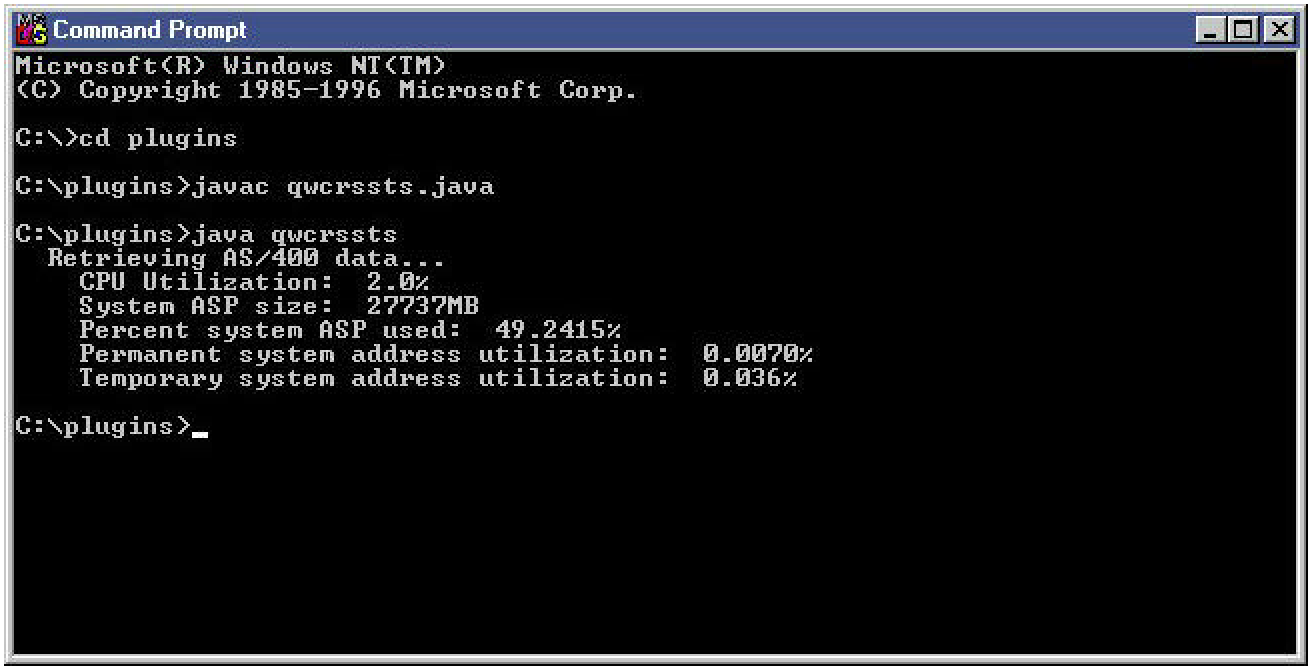
10. Run the test program from an MS-DOS prompt.

```
java qwcrssts
```

The following logon screen will appear. Sign-on to the iSeries using the userid and password you were given for the lab.



After logging on to the iSeries, the status data will be displayed in the MS-DOS window.



```
Microsoft(R) Windows NT(TM)
(C) Copyright 1985-1996 Microsoft Corp.

C:\>cd plugins

C:\plugins>javac qwcrssts.java

C:\plugins>java qwcrssts
  Retrieving AS/400 data...
    CPU Utilization:  2.0%
    System ASP size:  27737MB
    Percent system ASP used:  49.2415%
    Permanent system address utilization:  0.0070%
    Temporary system address utilization:  0.036%

C:\plugins>_
```

Didn't work? Now what?

•Check the classpath. Does it include everything that was listed in the setup portion of the lab? Did you run the javasetup batch file before you compiled the file?

•Is the the PCML file in a directory or JAR file that is in the CLASSPATH?

•Check qwcrssts.java for typing errors. Remember that Java is case-sensitive! Also check for semi-colons and curly braces.

•If you do not receive a logon screen to the iSeries, the test program is not actually calling the PCML program (the logon screen only appears when the program is called, not when the iSeries object is instantiated). Verify that the program name that you specified in the callProgram() method is the same name as the program name in the PCML file.

Remember to ask a lab attendant for help or consult Appendix A for the solutions at any time during this lab, if you get stuck.

Part 3: Integrate the PCML into the GUI application

1. Open the SystemStatusEngine.java file (from Exercise 2: Databeans) using the TextPad editor.
2. Add a new import statements.

   ```
   import com.ibm.as400.data.*;
   ```

   The com.ibm.as400.data package contains the classes to work with the PCML (ProgramCallDocument class).

3. Within the load() method, create a new object named value that will receive the iSeries data.

   ```
   Object value;
   ```

4. Replace the SystemStatus object with a ProgramCallDocument object.

   ```
   ProgramCallDocument pcmlDoc = new ProgramCallDocument(m_sAS400, "qwcrssts");
   ```

5. Call the callProgram() method on the ProgramCallDocument object. This method takes the program name (specified in the PCML file) as a parameter.

   ```
   pcmlDoc.callProgram("qwcrssts_SSTS0200");
   ```

6. Change the attribute assignments to retrieve the values from the ProgramCallDocument object using the getValue() method. First, using the value object, retrieve the value. Then, construct the output string and assign it to the attribute. For example, the CPU utilization attribute would change from:

   ```
   m_sCpuUtilization = new Float (aStatus.getPercentProcessingUnitUsed()).toString() + " %";
   ```

   To

   ```
   value = ((((Integer) pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.CpuUtilization")).floatValue())/10);
   m_sCpuUtilization = new String (value.toString() + " %");
   ```

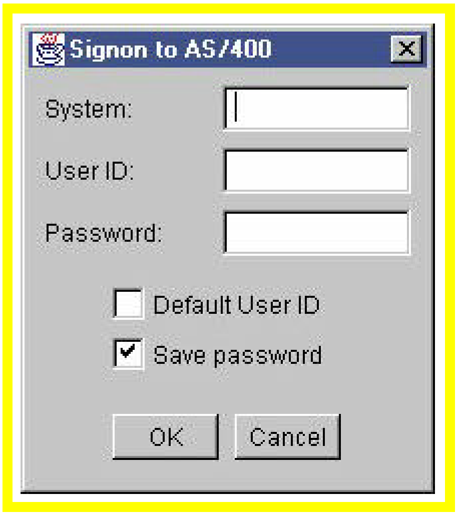
7. Compile the test program from an MS-DOS prompt.

   ```
   javac SystemStatusEngine.java
   ```

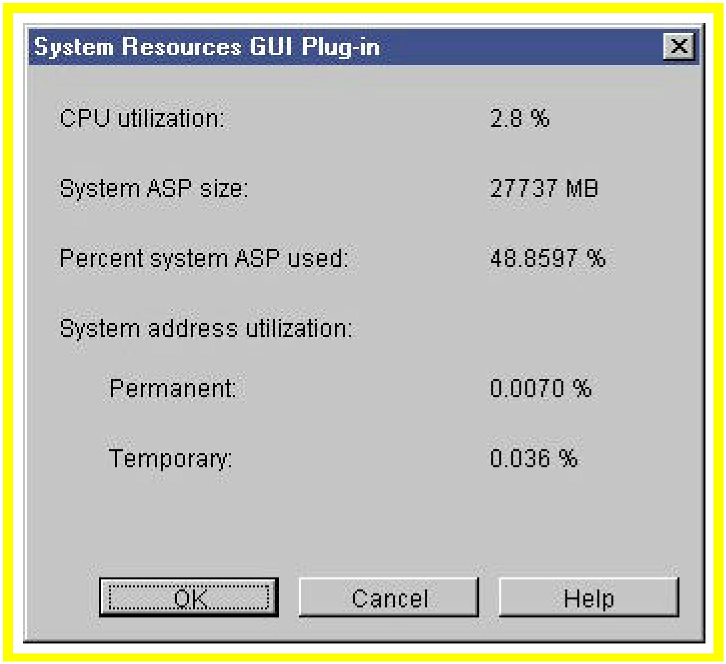
8. Run the SystemStatusTest program (created in Exercise 2: Databeans) from an MS-DOS prompt.

   ```
   java SystemStatusTest
   ```

   The following logon screen will appear. Sign-on to the iSeries using the userid and password you were given for the lab.



After logging on to the iSeries, the GUI panel will appear with the data retrieved from the PCML file.



CPU utilization:	2.8 %
System ASP size:	27737 MB
Percent system ASP used:	48.8597 %
System address utilization:	
Permanent:	0.0070 %
Temporary:	0.036 %

Didn't work? Now what?

•Check the classpath. Does it include everything that was listed in the setup portion of the lab? Did you run the javasetup batch file before you compiled the file?

•Is the the PCML file in a directory or JAR file that is in the CLASSPATH?

•Check SystemStatusEngine.java for typing errors. Remember that Java is case-sensitive! Also check for semi-colons and curly braces.

Conclusion

In this lab, you built a Java GUI plugin to Operations Navigator that gets and retrieves system status from the iSeries.

The following Redbooks are available at: http://www.redbooks.ibm.com/
1. AS/400 XML in Action: PDML and PCML , SG24-5959-00 (The entire book is available on the web in PDF format)
2. Building AS/400 Client/Server Applications with Java SG24-2152-02

On-line help for the GUI Builder and Resource Script Converter is available in the Info Center
- For Plug-in information via the Client Access Programming links
  http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/info/index.htm
- For the Graphical Toolbox and PCML via the Java and TOOLBOX FOR JAVA links
  http://publib.boulder.ibm.com/pubs/html/as400/v5r1/ic2924/info/java/rzahh/toolbox.htm
  Then take links for Graphical Toolbox or PCML

The Operations Navigator website at: http://www.as400.ibm.com/oper_nav/

You can get more information about the TOOLBOX FOR JAVA and download a trial or beta version by going to the web address http://www.ibm.com/as400/toolbox.

Appendix A: Solutions

Exercise 1: GUI Builder

SystemStatus.pdml

```
<PDML version="2.0" source="JAVA" basescreensize="1024x768">
        <PANEL name="SystemResources">
                <TITLE>SystemResources</TITLE>
                <SIZE>335,285</SIZE>
                <BUTTON name="BUTTON1">
                        <TITLE>SystemResources.BUTTON1</TITLE>
                        <LOCATION>35,255</LOCATION>
                        <SIZE>90,20</SIZE>
                        <ACTION>COMMIT</ACTION>
                        <HELPLINK>SystemResources.BUTTON1</HELPLINK>
                </BUTTON>
                <BUTTON name="BUTTON2">
                        <TITLE>SystemResources.BUTTON2</TITLE>
                        <LOCATION>135,255</LOCATION>
                        <SIZE>90,20</SIZE>
                        <ACTION>CANCEL</ACTION>
                        <HELPLINK>SystemResources.BUTTON2</HELPLINK>
                </BUTTON>
                <BUTTON name="BUTTON3">
                        <TITLE>SystemResources.BUTTON3</TITLE>
                        <LOCATION>235,255</LOCATION>
                        <SIZE>90,20</SIZE>
                        <ACTION>HELP</ACTION>
                        <HELPLINK>SystemResources.BUTTON3</HELPLINK>
                </BUTTON>
                <LABEL name="CPU_Label">
                        <TITLE>SystemResources.CPU_Label</TITLE>
                        <LOCATION>15,15</LOCATION>
                        <SIZE>195,20</SIZE>
                        <HELPLINK>SystemResources.CPU_Label</HELPLINK>
                </LABEL>
                <LABEL name="ASPSize_Label">
                        <TITLE>SystemResources.ASPSize_Label</TITLE>
                        <LOCATION>15,50</LOCATION>
                        <SIZE>195,20</SIZE>
                        <HELPLINK>SystemResources.ASPSize_Label</HELPLINK>
                </LABEL>
                <LABEL name="ASPUsed_Label">
                        <TITLE>SystemResources.ASPUsed_Label</TITLE>
                        <LOCATION>15,85</LOCATION>
                        <SIZE>195,20</SIZE>
                        <HELPLINK>SystemResources.ASPUsed_Label</HELPLINK>
                </LABEL>
                <LABEL name="Addresses_Label">
                        <TITLE>SystemResources.Addresses_Label</TITLE>
                        <LOCATION>15,120</LOCATION>
                        <SIZE>195,20</SIZE>
                        <HELPLINK>SystemResources.Addresses_Label</HELPLINK>
                </LABEL>
                <LABEL name="Perm_Label">
                        <TITLE>SystemResources.Perm_Label</TITLE>
                        <LOCATION>40,150</LOCATION>
                        <SIZE>165,20</SIZE>
                        <HELPLINK>SystemResources.Perm_Label</HELPLINK>
                </LABEL>
                <LABEL name="Temp_Label">
                        <TITLE>SystemResources.Temp_Label</TITLE>
                        <LOCATION>40,185</LOCATION>
                        <SIZE>165,20</SIZE>
                        <HELPLINK>SystemResources.Temp_Label</HELPLINK>
                </LABEL>
                <LABEL name="CPU_Value">
                        <TITLE>SystemResources.CPU_Value</TITLE>
                        <LOCATION>230,15</LOCATION>
                        <SIZE>90,20</SIZE>
                        <DATACLASS>SystemStatusEngine</DATACLASS>
                        <ATTRIBUTE>CpuUtilization</ATTRIBUTE>
                        <HELPALIAS>CPU_Label</HELPALIAS>
                </LABEL>
                <LABEL name="ASPSize_Value">
                        <TITLE>SystemResources.ASPSize_Value</TITLE>
                        <LOCATION>230,50</LOCATION>
                        <SIZE>90,20</SIZE>
                        <DATACLASS>SystemStatusEngine</DATACLASS>
                        <ATTRIBUTE>SystemASPSize</ATTRIBUTE>
                        <HELPALIAS>ASPSize_Label</HELPALIAS>
                </LABEL>
                <LABEL name="ASPUsed_Value">
                        <TITLE>SystemResources.ASPUsed_Value</TITLE>
                        <LOCATION>230,85</LOCATION>
                        <SIZE>90,20</SIZE>
                        <DATACLASS>SystemStatusEngine</DATACLASS>
                        <ATTRIBUTE>SystemASPPercent</ATTRIBUTE>
                        <HELPALIAS>ASPUsed_Label</HELPALIAS>
                </LABEL>
                <LABEL name="Perm_Value">
                        <TITLE>SystemResources.Perm_Value</TITLE>
                        <LOCATION>230,150</LOCATION>
                        <SIZE>90,20</SIZE>
                        <DATACLASS>SystemStatusEngine</DATACLASS>
                        <ATTRIBUTE>PermanentAddressesUsed</ATTRIBUTE>
                        <HELPALIAS>Perm_Label</HELPALIAS>
                </LABEL>
                <LABEL name="Temp_Value">
                        <TITLE>SystemResources.Temp_Value</TITLE>
                        <LOCATION>230,185</LOCATION>
                        <SIZE>90,20</SIZE>
                        <DATACLASS>SystemStatusEngine</DATACLASS>
```

```
                        <ATTRIBUTE>TemporaryAddressesUsed</ATTRIBUTE>
                        <HELPALIAS>Temp_Label</HELPALIAS>
                </LABEL>
        </PANEL>
</PDML>
```

Exercise 2: Data Beans

SystemStatusEngine.java

```
import com.ibm.as400.ui.framework.java.*;
import com.ibm.as400.access.*;
import com.ibm.as400.opnav.Monitor;

public class SystemStatusEngine extends Object
   implements DataBean
{
    private String m_sCpuUtilization;
    private String m_sSystemASPSize;
    private String m_sSystemASPPercent;
    private String m_sPermanentAddressesUsed;
    private String m_sTemporaryAddressesUsed;
   private AS400 m_sAS400;

   public SystemStatusEngine (AS400 anAS400)  {
      m_sAS400 = anAS400;
   }

    public String getCpuUtilization()
    {
       return m_sCpuUtilization;
    }

    public String getSystemASPSize()
    {
       return m_sSystemASPSize;
    }

    public String getSystemASPPercent()
    {
       return m_sSystemASPPercent;
    }

    public String getPermanentAddressesUsed()
    {
       return m_sPermanentAddressesUsed;
    }

    public String getTemporaryAddressesUsed()
    {
       return m_sTemporaryAddressesUsed;
    }

    public Capabilities getCapabilities()
    {
       return null;
    }
    public void verifyChanges()
    {
    }

    public void save()
    {
    }

    public void load()
    {
       try {
         SystemStatus aStatus = new SystemStatus(m_sAS400);
         m_sCpuUtilization = new Float (aStatus.getPercentProcessingUnitUsed()).toString() + " %";
         m_sSystemASPSize = new Integer (aStatus.getSystemASP()).toString() + " MB";
         m_sSystemASPPercent = new Float (aStatus.getPercentSystemASPUsed()).toString() + " %";
         m_sPermanentAddressesUsed = new Float (aStatus.getPercentPermanentAddresses()).toString() + " %";
         m_sTemporaryAddressesUsed = new Float (aStatus.getPercentTemporaryAddresses()).toString() + " %";
       } catch (Exception e)  {
         Monitor.logThrowable(e);
       }
    }
}
```

SystemStatusTester.java

```
import com.ibm.as400.access.*;
import com.ibm.as400.ui.framework.java.*;

class SystemStatusTester extends java.awt.Frame {

   public static void main (String[] args) {
      SystemStatusTester aSystemStatusTester = new SystemStatusTester();

      AS400 theMachine = new AS400();
      SystemStatusEngine theSystemEngine = new SystemStatusEngine(theMachine);
      theSystemEngine.load();

      DataBean[] dbeans = {theSystemEngine};
      PanelManager pm = null;
      try {
         pm = new PanelManager("SystemStatus", "SystemResources", dbeans, a
         SystemStatusTester);
      } catch (DisplayManagerException e) {
         e.displayUserMessage(aSystemStatusTester);
      }
      pm.setVisible(true);

   }

}
```

Exercise 3: Adding your plug-in to Operations Navigator

SystemStatusManager.java

```
import com.ibm.as400.opnav.*;
import com.ibm.as400.access.*;
import com.ibm.as400.ui.framework.java.*;
class SystemStatusManager extends Object implements ActionsManager {
private ObjectName[] initObjs;
private ObjectName dragDropObj;
public void initialize(ObjectName[] arg1, ObjectName arg2) {
initObjs = arg1;
dragDropObj = arg2;
}
public ActionDescriptor[] queryActions(int arg1) {
ActionDescriptor[] actions = new ActionDescriptor[0];
String objType = null;
try {
objType = initObjs[0].getObjectType();
if (objType.equals("AS4")) {
if ((arg1 & CUSTOM_ACTIONS) == CUSTOM_ACTIONS) {
actions = new ActionDescriptor[1];
ActionDescriptor act = new ActionDescriptor(1);
act.setText("System Status");
act.setHelpText("Loads the ITSO System Status Plugin");
act.setVerb("ITSOSysSts");
actions[0] = act;
}
}
}
catch (Exception e) {
Monitor.logThrowable(e);
}
return actions;
}
public void actionSelected(int arg1, java.awt.Frame arg2) {
if (arg1 == 1) {
try {
AS400 theMachine = (AS400)initObjs[0].getSystemObject();
SystemStatusEngine theSystemEngine = new SystemStatusEngine(theMachine);
theSystemEngine.load();
DataBean[] dbeans = {theSystemEngine};
PanelManager pm = null;
try {
pm = new PanelManager("SystemStatus","SystemResources",dbeans,arg2);
}
catch (DisplayManagerException e){
e.displayUserMessage(arg2);
}
pm.setVisible(true);
}
catch (Exception e) {
Monitor.logThrowable(e);
}
}
}
}
```

SysStat.reg

```
REGEDIT4
;
; Define the primary registry key for the plugin
;
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\LAB.SystemStatusPlugin]
;
; You have written an extension to the shell, so use EXT for the type of
; plugin. If a ListManager interface had been implemented then you would
; have set this to Plugin
;
"Type"="EXT"
"MinimumRISCRelease"="ANY"
"MinimumIMPIRelease"="NONE"
"ProductID"="NONE"
"ServerEntryPoint"="NONE"
; The next value point to the base directory or archive used to find any
; required user class definitions.
"JavaPath"="C:\\plugins\\solutions\\Exercise 3 -- ActionManager;"
"JavaMRI"=""
;
; Although no DLL is used you are required to add a registry entry. It will
; never be used or accessed but it MUST exist.
;
"NLS"="cwbunmri.dll"
"NameID"=dword:00000000
"DescriptionID"=dword:00000000
;--------------------------------------------------------------------
;Register a context menu handler for the new folder and its objects
;
[HKEY_CLASSES_ROOT\IBM.AS400.Network\3RD PARTY EXTENSIONS\LAB.SystemStatusPlugin\shellex\AS/400
Network\AS/400 System\ContextMenuHandlers\{1827A857-9C20-11d1-96C3-00062912C9B2}]
"JavaClass"="SystemStatusManager"
```

Exercise 4: Installing your Plug-in

Setup.ini

```
;--------------------------------------------------------------------
;
; File:
;   SETUP.INI
;
; Purpose:
;   Provide the information needed to install, upgrade, and service
;   an Operations Navigator plug-in.  The information provided in
;   this file and the MRISETUP.INI file will allow the Client Access
;   Selective Setup program to install this plug-in.  It will
;   also provide the information needed by the Client Access Login
;   Service Check program to determine when the plug-in needs to be
;   upgraded or serviced on the customer's computer.
;
; Usage Notes:
;   1. This file as well as MRISETUP.INI must be present when the code
;      for your plug-in is installed on the AS/400.
;
;   2. This file should be an ASCII flat file with each line terminated
;      with both a carriage return and a line feed character.
;
;   3. See further usage notes below for each of the sections
;      defined in this file.
;
; For information on how to implement Operations Navigator plug-ins,
; browse the AS/400 Information Center and Technical Studio web sites
; at http://www.as400.ibm.com/infocenter.
;
;--------------------------------------------------------------------
; The Plugin Info section provides the following information to the
; Client Access Selective Setup and Login Service Check programs:
;
; Name              - English name of the plug-in.  This name
;                     will be displayed during installation of
;                     the plug-in if the translated name cannot
;                     be determined.
;
; NameDLL           - Name of the resource DLL located in the plug-in's
;                     MRI directories containing the translated name of
;                     the plug-in.
;
; NameResID         - Resource ID of the translated name in the
;                     MRI DLL.
;
; Description       - English description of the plug-in.  This
;                     description will be displayed during installation of
;                     the plug-in if the translated description cannot
;                     be determined.
;
; DescriptionDLL    - Name of the resource DLL located in the plug-in's
;                 MRI directories containing the translated
;                 description of the plug-in
;
; DescriptionResID   - Resource ID of the translated description in
;                 the MRI DLL.
;
; Version          - A numeric value indicating the release level of the
;                 plug-in.  This value is used by Client Access
;                 Login Service Check program to determine whether
;                 the plug-in requires an Upgrade on the customer's
;                 computer. This value must be incremented by some
;                 amount for each new release of the plug-in.  The
;                 Version value is compared to the current Version
;                 value of the installed plug-in on the customer's
;                 computer.  If this Version value is greater than
;                 that of the plug-in installed on the customer's
;                 computer the plug-in will be Upgraded to the new
;                 Version by the Client Access Login Service Check
;                 program.
;
; VendorID         - A Vendor.Component value that is used to identify
;                 the plug-in. This VendorID will be used to create
;                 the plug-in's registry key in the Client Access
;                 registry tree.  The Vendor.Component name must
;                 be identical to the Vendor.Component portion of the
;                 path where the plug-in will be installed on the
;                 AS/400.
;
; SupportExpress     - The value of YES is required for plug-ins that
;                 are designed for Java Plug-ins
;-------------------------------------------------------------------

[Plugin Info]
Name=System Status Sample
NameDLL=SysStat.dll
NameResID=1
Description=Operations Navigator Java Context Menu Plug-in for System Status
DescriptionDLL=SysStat.dll
DescriptionResID=2
Version=0
VendorID=LAB.SystemStatusPlugin
SupportExpress=YES


;-------------------------------------------------------------------
; The Service section provides the following information to the
; Client Access Selective Setup and Login Service Check programs:
;
; Fixlevel         - A numeric value indicating the service level of the
;                 plug-in.  This value is used by the Client Access
;                 Login Service Check program to determine whether
;                 the plug-in requires servicing. This value must be
;                 incremented by some amount with each service release
;                 for a particular Version.  The FixLevel value is
;                compared to the current FixLevel value of the installed
;                plug-in on the customer's computer.  If this
;                FixLevel value is greater than that of the plug-in
;                installed on the customer's computer the plug-in
;                will be Serviced to the new FixLevel by the Client
;                Access Login Service Check program. This value must
;                be reset to zero when an plug-in is upgraded to a new
;                Version or release level.
;
; AdditionalSize   - The amount of DASD space required to store any new
;                or additional executable files added to the
;                plug-in during servicing.  This value is
;                used by Install to determine if the PC has adequate
;                disk space for the plug-in.
;-------------------------------------------------------------------

[Service]
FixLevel=0
AdditionalSize=0



;-------------------------------------------------------------------
; This following sections list the files to be installed for the
; plug-in.  The file section in which a file is listed is used
; by Client Access Selective Setup to determine the file's source and
; target location.
;
; The file sections used during the initial install or during an
; upgrade to a new Version or release level are:
;
; Base Files       - Files copied to \Plugins\Vendor.Component under the
;                Client Access install directory.  For example
;                C:\Program Files\IBM\Client Access\Plugins\IBM.JavaSample
;
; Shared Files     - Files copied to the Client Access Shared
;                directory.  For example
;                C:\Program Files\IBM\Client Access\Shared
;
; System Files     - Files copied to the \Windows\System or
;                \WinNT\System32 directory.
;
; Core Files       - File copied to the \Windows\System or
;                \WinNT\System32 directory that are usecounted
;                in the registry and are never removed.  These
;                are typically redistributable files.
;
; MRI Files        - Files that are copied from the plug-in's MRI
;                directories on the AS/400 to the Client
;                Access\MRI29xx\Vendor.Component directories on the
;                PC. For example
;                C:\Program Files\IBM\Client Access\MRI2924\IBM.JavaSample
;
; Java MRI Files   - Files that are copied from the plug-in's MRI
;                directories on the AS/400 to the same directory
;                  to which the [Base Files] are installed. This typically
;                  is where Java translatabe 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 copied from the plug-in's MRI
;                  directories on the AS/400 to the
;                  Client Access\MRI29xx\Vendor.Component directories
;                  on the PC. For example
;                  C:\Program Files\IBM\Client Access\MRI2924\IBM.JavaSample
;                  Help registry information is also written for these
;                  files to
;                  HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\Help
;
; Registry Files     - Exported Regedit files to be processed.
;
; Dependencies       - Defines the subcomponents 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 subcomponents to be installed
;                  besides the Operations Navigator base support subcomponent.  Java plug-ins
;                  should always specify, at a minimum, that they require the AS/400 Java Toolbox.
;
;                  Two values are supported:
;
;                    AS400_Operations_Navigator
;                      This value is used for legacy purposes to identify the
;                      subcomponents that must be installed if the plug-in is installed
;                      on Client Access V3R2M0. If the plug-in does not support
;                      running on Client Access V3R2M0, this value should not be
;                      specified.
;                      The subcomponents are specified in a comma-delimited list. A
;                      single subcomponent is specified as a single number
;                      (AS400_Operations_Navigator=3). The CWBUN.H header
;                      file contains a list of constants that are prefixed with
;                      CWBUN_OPNAV_. These constants provide the numeric
;                      values that are used in the comma-delimited list for
;                      AS400_Operations_Navigator.
;
;                    AS400_Client_Access_Express
;                      This value is used to identify the subcomponents that must be
;                      installed if the plug-in is installed on Client Access Express.
;                      The subcomponents are specified in a comma-delimited list. A
;                      single subcomponent is specified as a single number
;                      (AS400_Client_Access_Express=3). The CWBAD.H header
;                      file contains a list of constants that 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
;                      subcomponents. These constants must not be used in the
;                      AS400_Client_Access_Express value and are listed below:
;
;                        //5250 Display and Printer Emulator subcomponents
;                    #define CWBAD_COMP_PC5250_BASE_KOREAN     (150)
;                    #define CWBAD_COMP_PC5250_PDFPDT_KOREAN   (151)
;                    #define CWBAD_COMP_PC5250_BASE_SIMPCHIN   (152)
;                    #define CWBAD_COMP_PC5250_PDFPDT_SIMPCHIN (153)
;                    #define CWBAD_COMP_PC5250_BASE_TRADCHIN   (154)
;                    #define CWBAD_COMP_PC5250_PDFPDT_TRADCHIN (155)
;                    #define CWBAD_COMP_PC5250_BASE_STANDARD   (156)
;                    #define CWBAD_COMP_PC5250_PDFPDT_STANDARD (157)
;                    #define CWBAD_COMP_PC5250_FONT_ARABIC     (158)
;                    #define CWBAD_COMP_PC5250_FONT_BALTIC     (159)
;                    #define CWBAD_COMP_PC5250_FONT_LATIN2     (160)
;                    #define CWBAD_COMP_PC5250_FONT_CYRILLIC   (161)
;                    #define CWBAD_COMP_PC5250_FONT_GREEK      (162)
;                    #define CWBAD_COMP_PC5250_FONT_HEBREW     (163)
;                    #define CWBAD_COMP_PC5250_FONT_LAO        (164)
;                    #define CWBAD_COMP_PC5250_FONT_THAI       (165)
;                    #define CWBAD_COMP_PC5250_FONT_TURKISH    (166)
;                    #define CWBAD_COMP_PC5250_FONT_VIET       (167)
;
;              This value is ignored by Client Access V3R2M0.
;
;           Note:
;             Client Access Express will use the
;             AS400_Client_Access_Express value if it exists. If it does
;             not exist, it will use the AS400_Operations_Navigator value,
;             if it exists. If neither value exists, then this section
;             (Dependencies) is ignored.
;
;
; The file sections used when servicing an plug-in when the FixLevel
; value is incremented are listed below.  Note that MRI and Help files
; cannot be serviced.
;
; Service Base Files     - Files copied to \Plugins\Vendor.Component                under the Client Access
directory.
;                 under the Client Access install directory.
;                 For example
;                 C:\Program Files\IBM\Client Access\Plugins\IBM.JavaSample
;
; Service Shared Files    - Files copied to Client Access Shared
;                 directory.  For example
;                 C:\Program Files\IBM\Client Access\Shared
;
; Service System Files    - Files copied to the \Windows\System or
;                 \WinNT\System32 directory.
;
; Service Core Files     - File copied to the \Windows\System or
;                 \WinNT\System32 directory that are usecounted
;                 in the registry and are never removed. These
;                 are typically redistributable files.
;
; Service Registry Files   - Exported Regedit files to be processed.
;
;
```

```
; The files listed in each file section below should be listed as
;
; n=file.ext
;
; where "n" is the number of the file in that section.  The number "n"
; of the files must start with one and increment by one until
; all of the files are listed in the section.  For example
;
; [Base Files]
; 1=file1.dll
; 2=file2.dll
; 3=file3.dll
;
; Only file names and plug-in are supported.  Do not specify path
; names as part of file name and plug-in.
;
; File sections can be empty.  Those file sections are simply ignored.
;--------------------------------------------------------------------

;--------------------------------------------------------------------
; The Base Files section
;--------------------------------------------------------------------

[Base Files]
1=SystemStatusTester.class
2=SystemStatusEngine.class
3=SystemStatusManager.class
;
;Note: instead of individual files, you can create a .jar file and list it here
;1=javasample.jar


;--------------------------------------------------------------------
; The System Files section
;--------------------------------------------------------------------

[System Files]


;--------------------------------------------------------------------
; The Shared Files section
;--------------------------------------------------------------------

[Shared Files]


;--------------------------------------------------------------------
; The Core Files section
;--------------------------------------------------------------------

[Core Files]


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

```
; The MRI Files section
;--------------------------------------------------------------------

[MRI Files]
1=SysStat.dll


;--------------------------------------------------------------------
; The Java MRI Files section (example for Canadian French)
;--------------------------------------------------------------------

[JavaMRI2981]
;for example
;1=javasamplemri_fr_CA.jar

;--------------------------------------------------------------------
; The Help Files section
;--------------------------------------------------------------------

[Help Files]


;--------------------------------------------------------------------
; The Registry Files section
;--------------------------------------------------------------------

[Registry Files]
1=SysStat.reg


;--------------------------------------------------------------------
; The Install Dependencies section - The AS/400 Java Toolbox is listed
; as a dependency below.
;--------------------------------------------------------------------
[Dependencies]
AS400_Client_Access_Express=16


;--------------------------------------------------------------------
; The Service Base Files section
;--------------------------------------------------------------------

[Service Base Files]


;--------------------------------------------------------------------
; The Service System Files section
;--------------------------------------------------------------------

[Service System Files]


;--------------------------------------------------------------------
; The Service Shared Files section
```

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

[Service Shared Files]


;---------------------------------------------------------------------
; The Service Core Files section
;---------------------------------------------------------------------

[Service Core Files]


;---------------------------------------------------------------------
; The Service Registry Files section
;---------------------------------------------------------------------

[Service Registry Files]
```

Exercise 5: PCML

qwcrssts.pcml

```
<pcml version="1.0">

<struct name="SSTS0200">
  <data                                   type="byte"    length="32"    />
  <data name="CpuUtilization"             type="int"     length="4"     />
  <data                                   type="byte"    length="4"     />
  <data name="PermanentAddressesUsed"     type="int"     length="4"     />
  <data name="TemporaryAddressesUsed"     type="int"     length="4"     />
  <data name="SystemASPSize"              type="int"     length="4"     />
  <data name="SystemASPPercent"           type="int"     length="4"     />
  <data                                   type="byte"    length="12"    />
</struct>

<program name="qwcrssts_SSTS0200"         path="/QSYS.LIB/QWCRSSTS.PGM">
  <data name="receiver"           type="struct"    usage="output"   struct="SSTS0200"/>
  <data name="receiverLength"     type="int"       length="4"       usage="input"  init="68"/>
  <data name="format"          type="char"         length="8"       usage="input"  init="SSTS0200"/>
  <data name="resetStatus"        type="char"      length="10"      usage="input"  init="*NO" />
  <data name="errorCode"        type="int"         length="4"       usage="input"  init="0"/>
</program>

</pcml>
```

qwcrssts.java

```
import com.ibm.as400.access.*;
import com.ibm.as400.data.*;

public class qwcrssts
{

  public static void main(String[] argv)
  {
    Object value;
    AS400 theMachine = new AS400();

    try {
        ProgramCallDocument pcmlDoc = new ProgramCallDocument(theMachine, "qwcrssts");
            pcmlDoc.callProgram("qwcrssts_SSTS0200");

            System.out.println("  Retrieving AS/400 data...");

            value = new Float ((((Integer)
pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.CpuUtilization")).floatValue())/10);
            System.out.println("    CPU Utilization:  " + value + "%");

            value = ((Integer) pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.SystemASPSize"));
            System.out.println("    System ASP size:  " + value + "MB");

            value = new Float ((((Integer)
pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.SystemASPPercent")).floatValue())/10000);
            System.out.println("    Percent system ASP used:  " + value + "%");

            value = new Float ((((Integer)
pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.PermanentAddressesUsed")).floatValue())/1000);
            System.out.println("    Permanent system address utilization:  " + value + "%");

            value = new Float ((((Integer)
pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.TemporaryAddressesUsed")).floatValue())/1000);
            System.out.println("    Temporary system address utilization:  " + value + "%");

    } catch (PcmlException e)  {
        System.out.println(e.getLocalizedMessage());
        e.printStackTrace();
        System.out.println("*** Call to QWCRSSTS failed. ***");
        System.exit(0);
    }
    System.exit(0);
  }
}
```

SystemStatusEngine.java

```
import com.ibm.as400.ui.framework.java.*;
import com.ibm.as400.access.*;
import com.ibm.as400.opnav.Monitor;
import com.ibm.as400.data.*;

public class SystemStatusEngine extends Object
   implements DataBean
{
   private String m_sCpuUtilization;
   private String m_sSystemASPSize;
   private String m_sSystemASPPercent;
   private String m_sPermanentAddressesUsed;
   private String m_sTemporaryAddressesUsed;
   private AS400  m_sAS400;

   public SystemStatusEngine (AS400 anAS400)  {
            m_sAS400 = anAS400;
  }

   public String getCpuUtilization()
   {
      return m_sCpuUtilization;
   }

   public String getSystemASPSize()
   {
      return m_sSystemASPSize;
   }

   public String getSystemASPPercent()
   {
      return m_sSystemASPPercent;
   }

   public String getPermanentAddressesUsed()
   {
      return m_sPermanentAddressesUsed;
   }

   public String getTemporaryAddressesUsed()
   {
      return m_sTemporaryAddressesUsed;
   }

   public Capabilities getCapabilities()
   {
      return null;
   }

   public void verifyChanges()
   {
   }
   public void save()
   {
   }

   public void load()
   {
            Object value;
      try {
        ProgramCallDocument pcmlDoc = new ProgramCallDocument(m_sAS400, "qwcrssts");
              pcmlDoc.callProgram("qwcrssts_SSTS0200");

              value = new Float ((((Integer)
pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.CpuUtilization")).floatValue())/10);
              m_sCpuUtilization = new String (value.toString() + " %");

              value = ((Integer) pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.SystemASPSize"));
        m_sSystemASPSize = new String (value.toString() + " MB");

              value = new Float ((((Integer)
pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.SystemASPPercent")).floatValue())/10000);
        m_sSystemASPPercent = new String (value.toString() + " %");

              value = new Float ((((Integer)
pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.PermanentAddressesUsed")).floatValue())/1000);
        m_sPermanentAddressesUsed = new String (value.toString() + " %");

              value = new Float ((((Integer)
pcmlDoc.getValue("qwcrssts_SSTS0200.receiver.TemporaryAddressesUsed")).floatValue())/1000);
        m_sTemporaryAddressesUsed = new String (value.toString() + " %");

      } catch (Exception e)  {
        Monitor.logThrowable(e);
      }
   }
}
```