AS/400e

Operations Navigator Plug-in Development

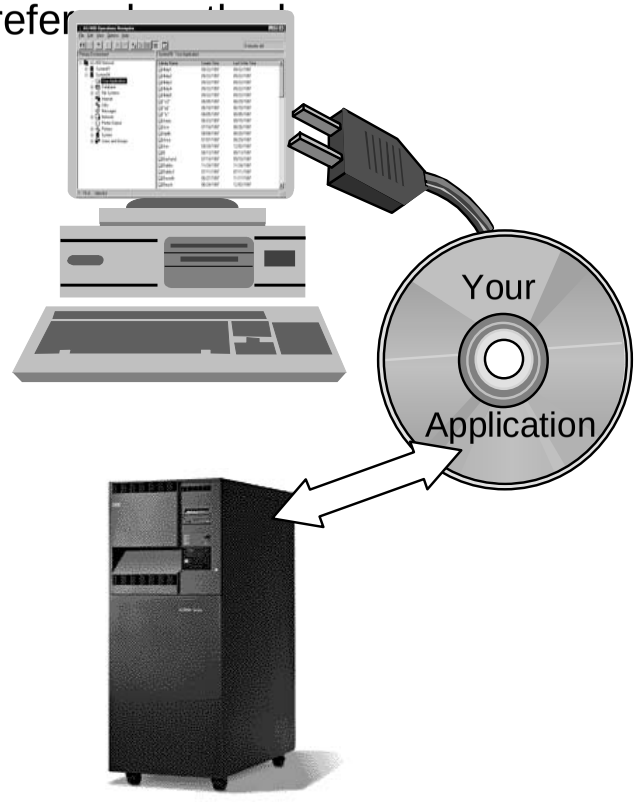
Scott Sylvester
AS/400 Software Development

Overview

- Op Nav Plug-in Support Capabilities
- Extending Operations Navigator
- Plug-in Concepts
- Plug-in Interfaces
- Development and Tools
  - PDML
  - Creating Java Windows
  - GUIBuilder
- Simplifying AS/400 Data Access
  - AS/400 Toolbox for Java
  - PCML - Simplifying AS/400 program calls
- Developing Java applications / plugins
- Plug-in Installation
- Problem determination
- Where to Get More Information

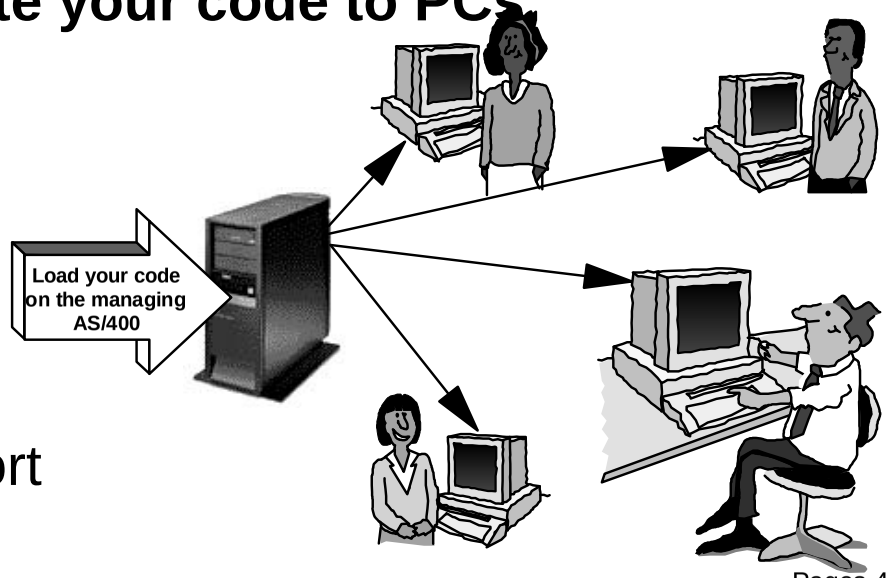
Plug-in Support Gives You the Capability to:

- **Plug in custom tools and applications**
  - Add new folders and objects to the Op Nav hierarchy, each having their own context menus, property pages and toolbars
- **MODIFY EXISTING Op Nav folders and objects by adding context menus to objects in the Op Nav hierarchy**
  - You can launch applications from a folder in the Navigator hierarchy
  - You can create new dialogs for existing objects in Operations Navigator
  - You can provide customized toolbars
- **Add new pages to the properites panel for a folder or object**
  - Gives users more ways to customize and control their AS/400 environment
  - Currently available only for C++ plugins
- **Helps you manage your applications**
  - installing fixes and code upgrades
  - uninstall
  - Provides multi-national language support
- **Plug-in Support for Java, C++, VisualBasic**
  - This presentation focuses on Java as the prefer



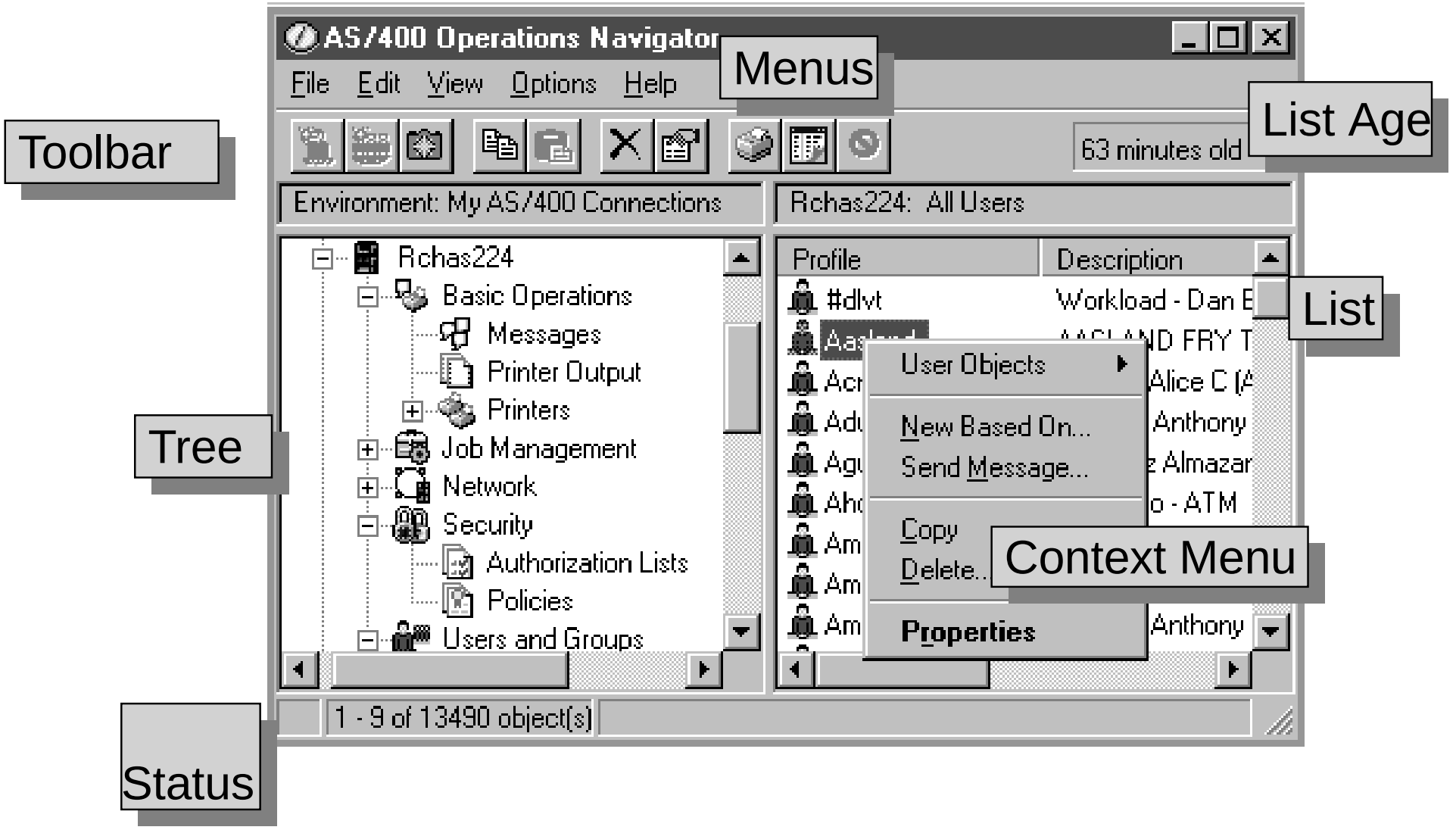
Benefits of Op Nav Plug-in Support:

- **Helps you to get your graphical application into the Op Nav tree**
- **Your function is accessible via Operations Navigator**
- **You can build graphical interfaces and hook them into the AS/400 graphical user interface**
- **Make your applications part of the direction of the AS/400**
- **Solves the problem of how to distribute your code to PCs**
- **Helps you manage your applications**
  - installing fixes and code upgrades
  - uninstall
  - Provides multi-national language support



Extending Operations Navigator

OpNav User Interface Elements



User Interface Characteristics

- Functions are arranged in a hierarchy
- User works with lists of objects that represent AS/400 resources
- Lists are "snapshots" of server activity
- Actions are defined on each object type
- Actions may be unique or common (e.g. Copy, Paste, Delete, Properties)
- Actions may be enabled or disabled based on the object's state
- Online help is available for each action

User Interface Characteristics (cont.)

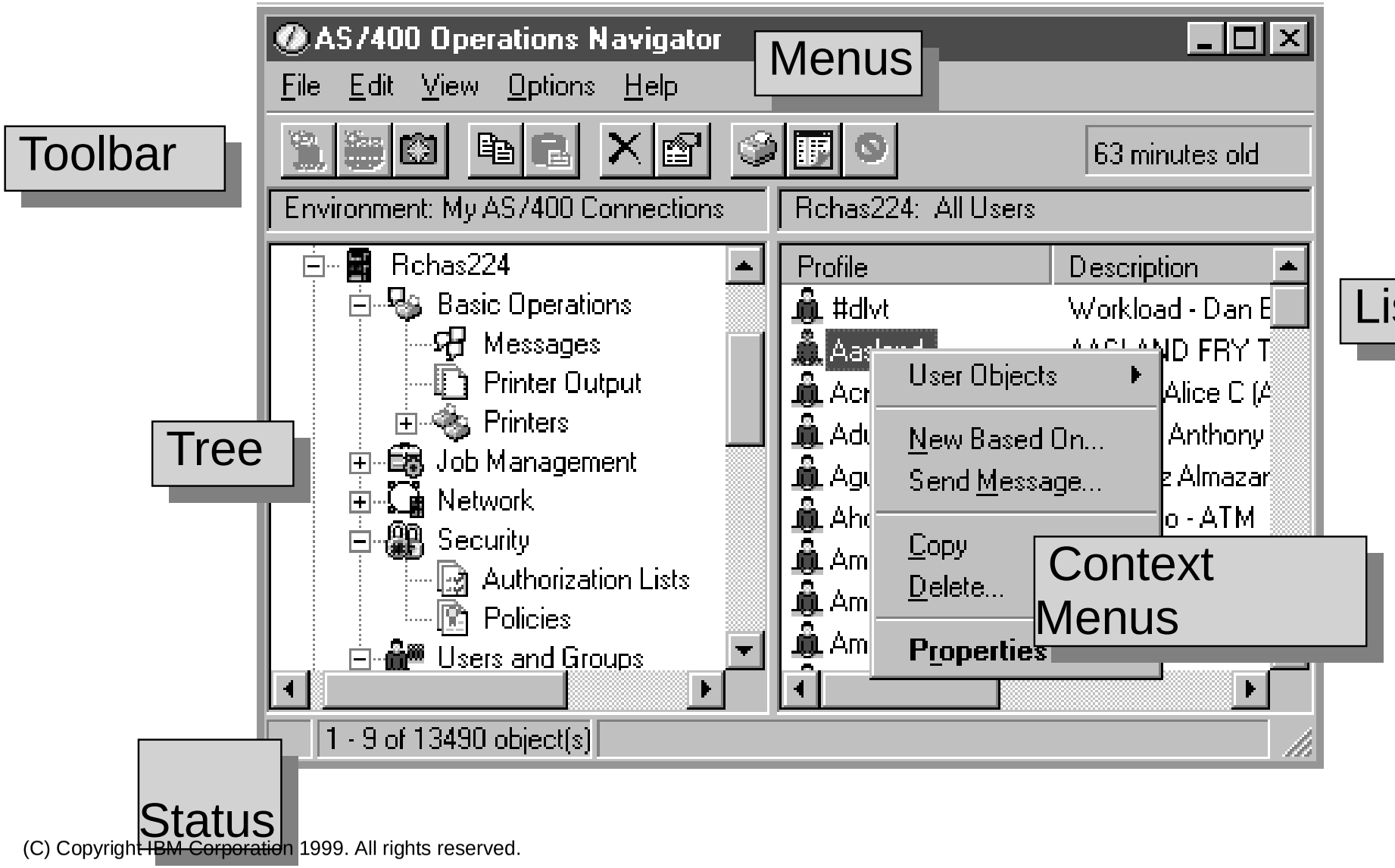
- When the user selects an action, Operations Navigator displays a dialog.
  - confirmation
  - single panel that contains additional data
  - property sheet or wizard
- When the user clicks OK, the requested action is performed.
  - If action was successful, the main OpNav window and status area are refreshed.
  - If an error occurred, an error message is displayed.
  - Return is made to the dialog to allow user to correct the error.

Extending OpNav with Plug-ins

- Operations Navigator is shipped as part of Client Access Express at no additional charge.
- AS/400 Toolbox for Java is API's, GUIs and tools for connecting to AS/400 via Java and is shipped with Client Access Express
- Business Partners may provide **Plug-ins**
  - set of resources, classes, interfaces, and other "cross-references" that allow OpNav to be extended without changing code
  - Allows third parties without knowing internals
- Plug-ins can be C++, Java or VisualBasic

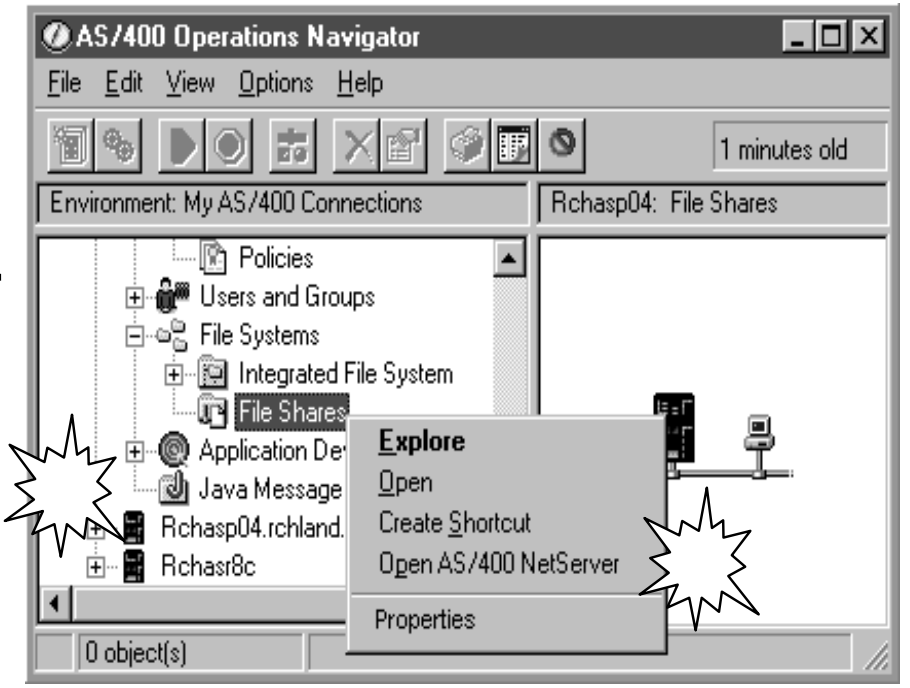
★NOTE: This presentation focuses on Java plug-ins

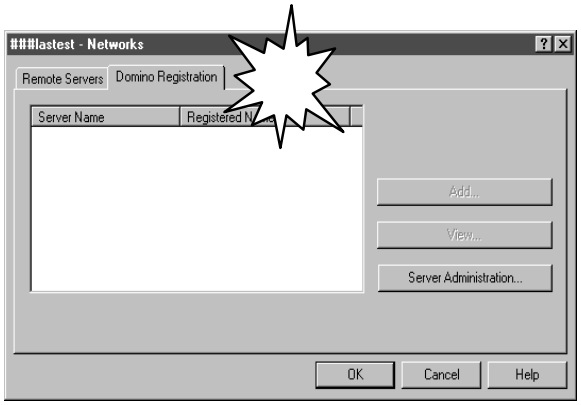
Op Nav Organization: Where can I fit in?



Attaching to the Tree

- Add your folder to the tree
  - At any container
- Add your action to another object's menu
- Add your property page to another object's property sheet. (C++ only at this time)

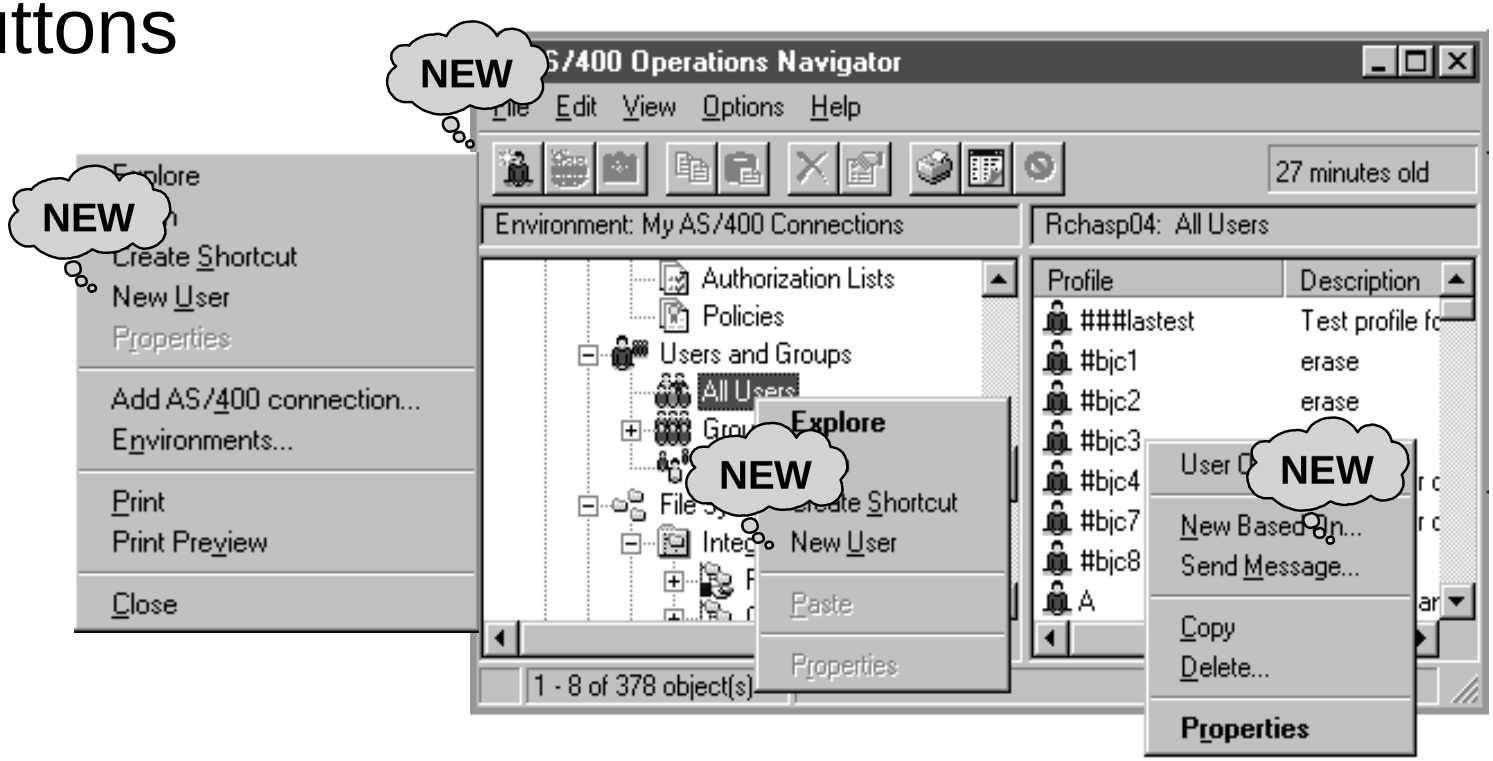




Menus and Buttons

Menu Items drive

- Pull down menus
- Context menus
- Buttons

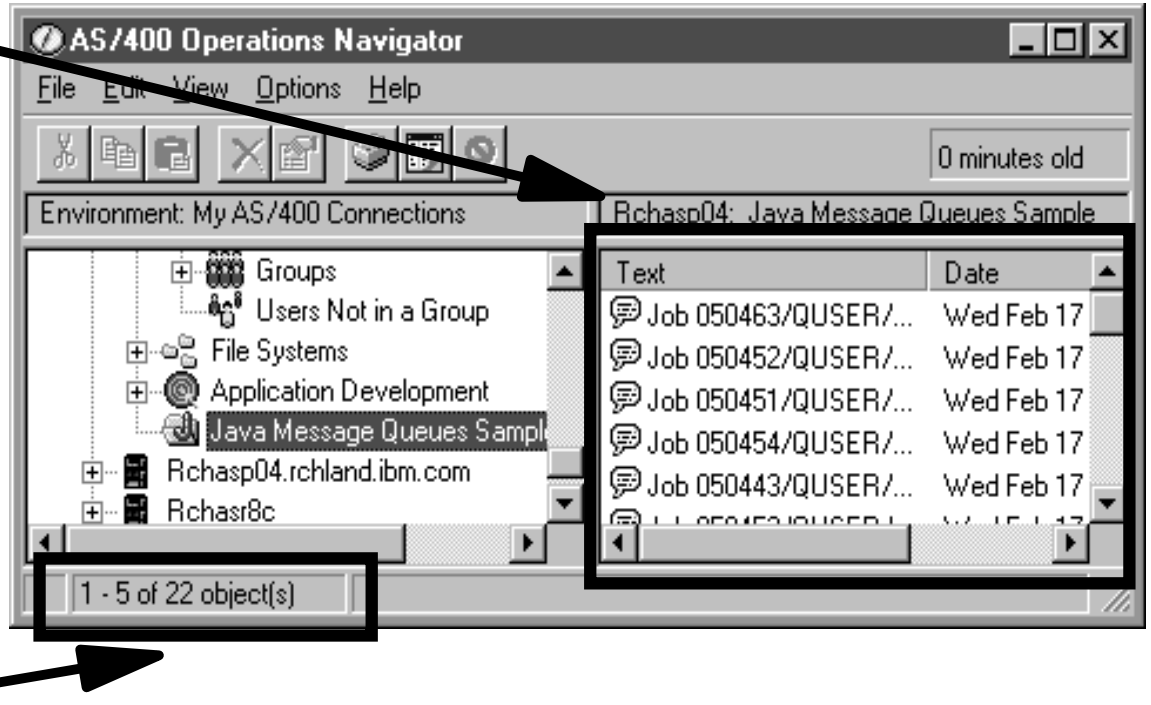


Menus and Buttons *(continued)*

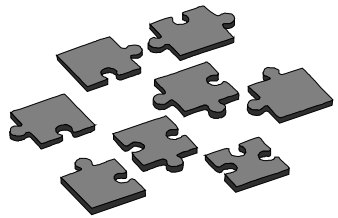
- Depends on currently selected item
- May depend on other things
  - Security etc.
- Each has associated action or command to invoke
  - Commands may be reused
- Pull-down and context menus have same content
  - same menu handler called with different flags
- Toolbar buttons also utilize menu handler

Lists and Status

- Enumerate objects
- Manage objects
- Manage columns
- Refresh levels
  - Object
  - List
  - Tree (everything)
- Status messages

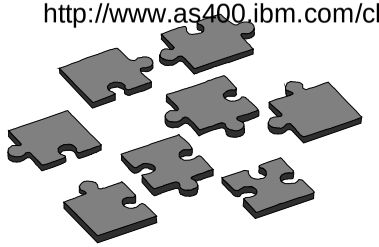


OpNav Building Blocks



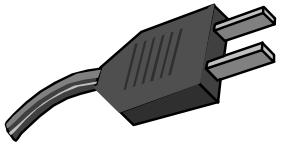
- Interfaces
  - actionManager
  - listManager
  - dropTarget
  - data server
  - property sheets
- Resources
  - MRI dll
- Classes
  - In jar files or dlls

OpNav Building Blocks *(continued)*

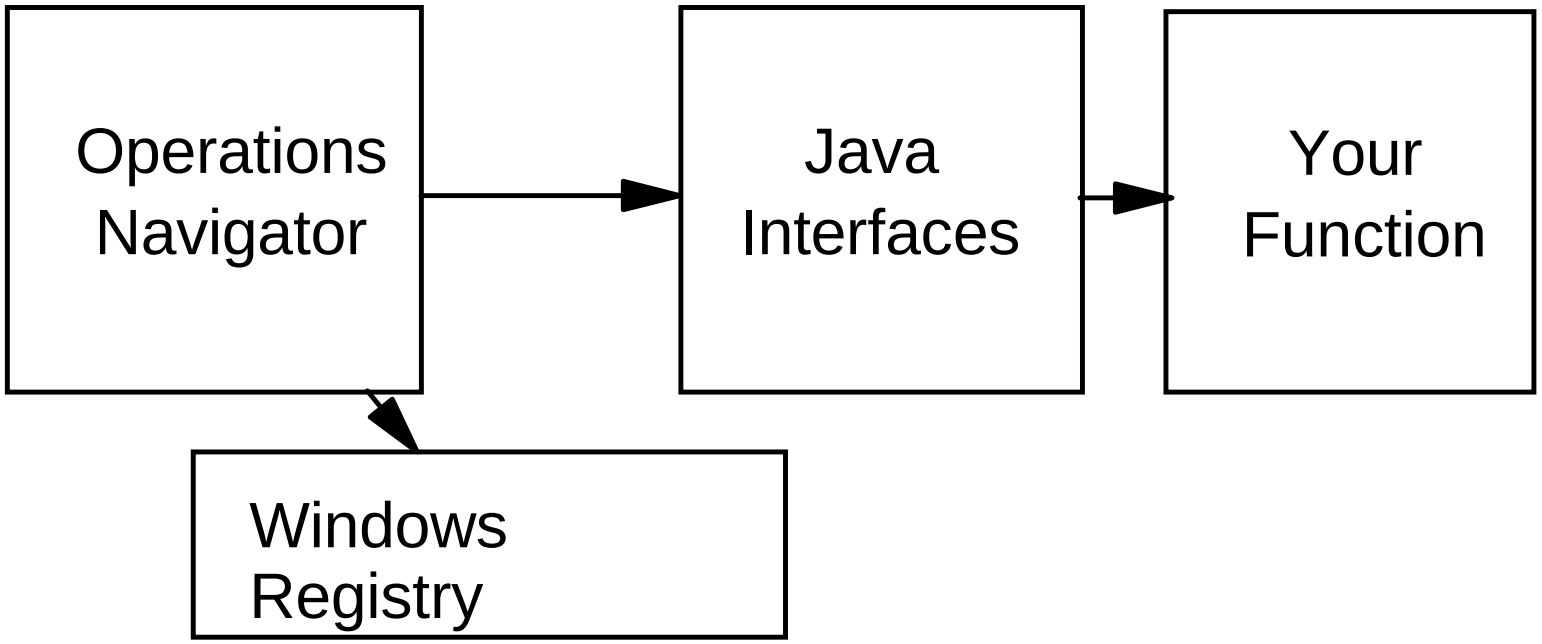


- Registry Entries
  - Provide "cross references"
- Common services
  - Refresh item / list
  - Set status text
  - Flyover help

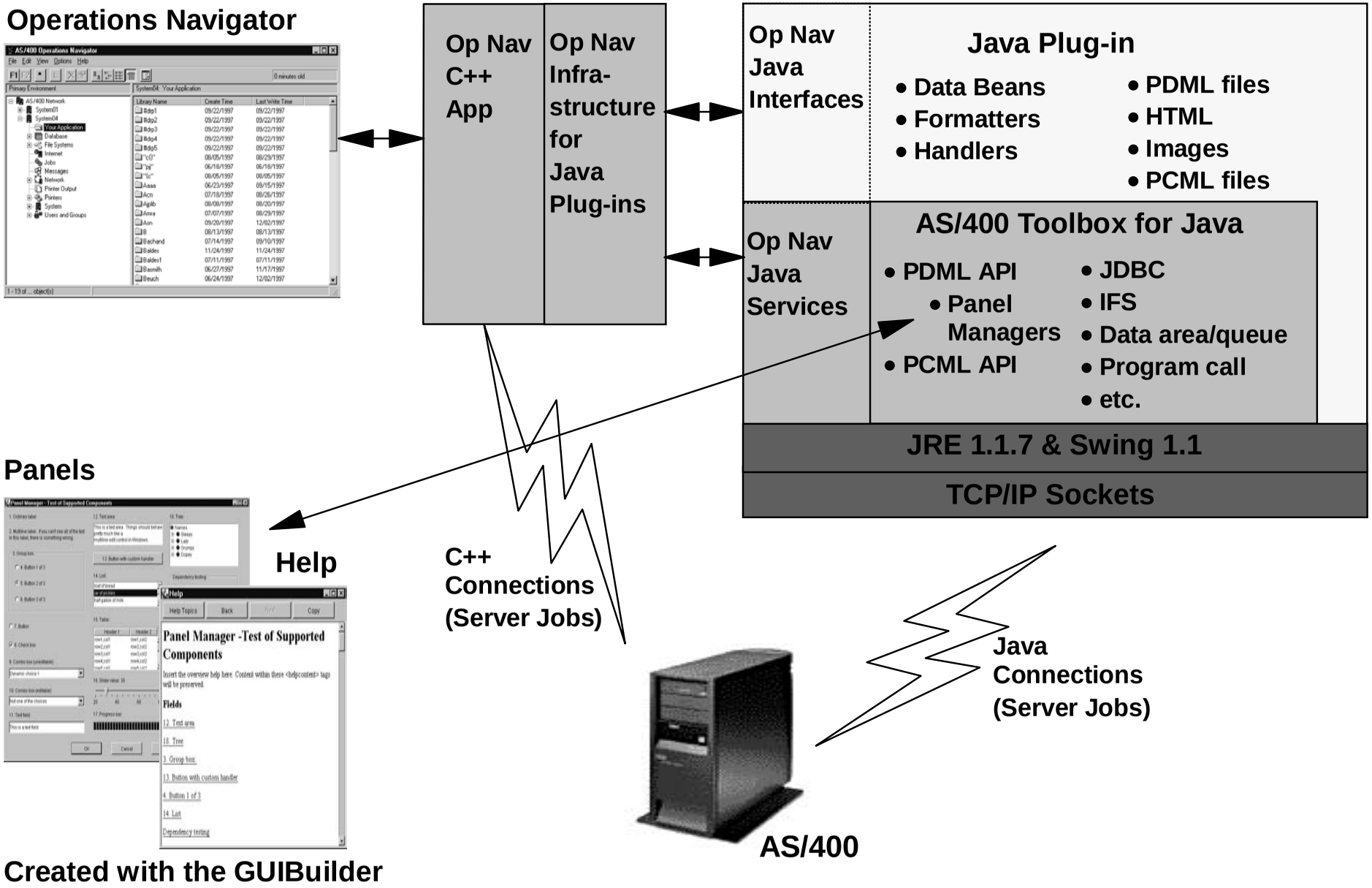
Plugging in your Plug-in



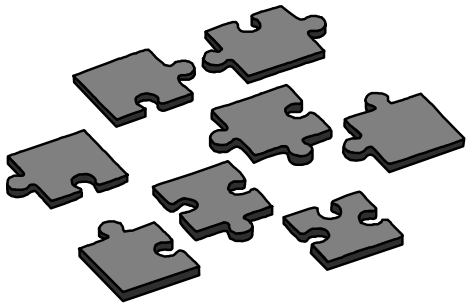
You use standard interfaces and provide Windows registry entries



Operations Navigator with Java Plug-ins

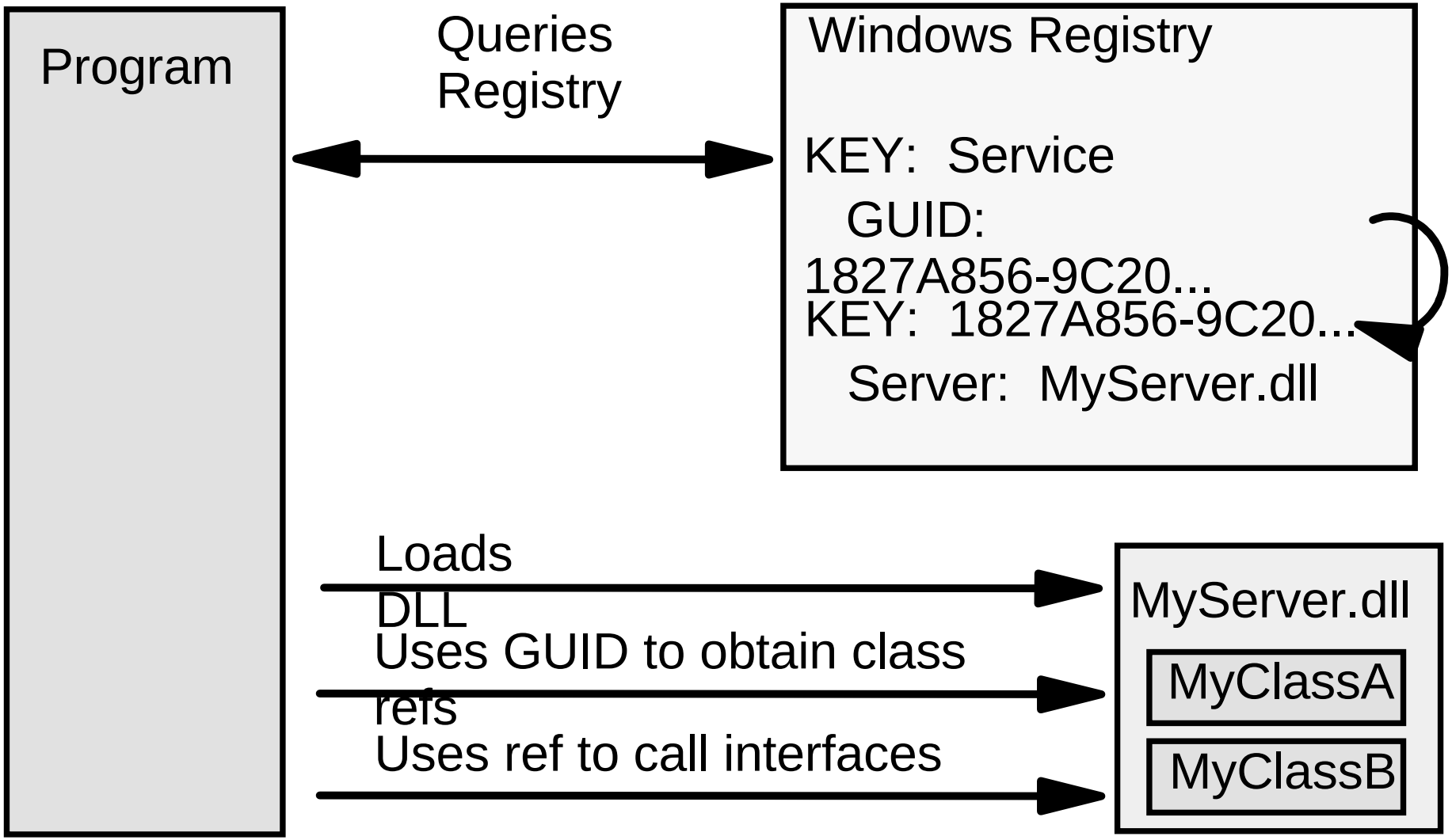


OpNav Plugin Foundation



- Based on Microsoft GUI Extendability and ActiveX
- aka. OLE II, COM (Component Object Model)
- GUID's - identifiers of classes
- Registry entries - to cross reference server to dll

Active X Diagram



Plug-in interfaces

OpNav Registry Entries

Interface types

- Context menu handlers
- Drag drop hander
- Property sheet handlers
- Data servers / List managers

Standard Interfaces

Interfaces parallel those used in C++

Function	Java Interface	C++ COM Interface
Context Menu Display and Command Execution	ActionsManager queryActions actionSelected	IContextMenu QueryContextMenu InvokeCommand
Enumerating Object Lists	ListManager open getItemCount itemAt getAttributes getColumnData ...etc	A4HierarchyFolder Open GetItemCount ItemAt GetAttributesOf GetColumnDataItem ...
Handling Drag and Drop	DropTargetManager dragEnter dragOver dragExit	IDropTarget DragEnter DragOver DragLeave

ActionsManager Interface

Display menu choices and act on menu requests

- initialize - passes in the object selected
- queryActions - lets you add menu items for the object
- actionSelected - tells you which menu item was selected

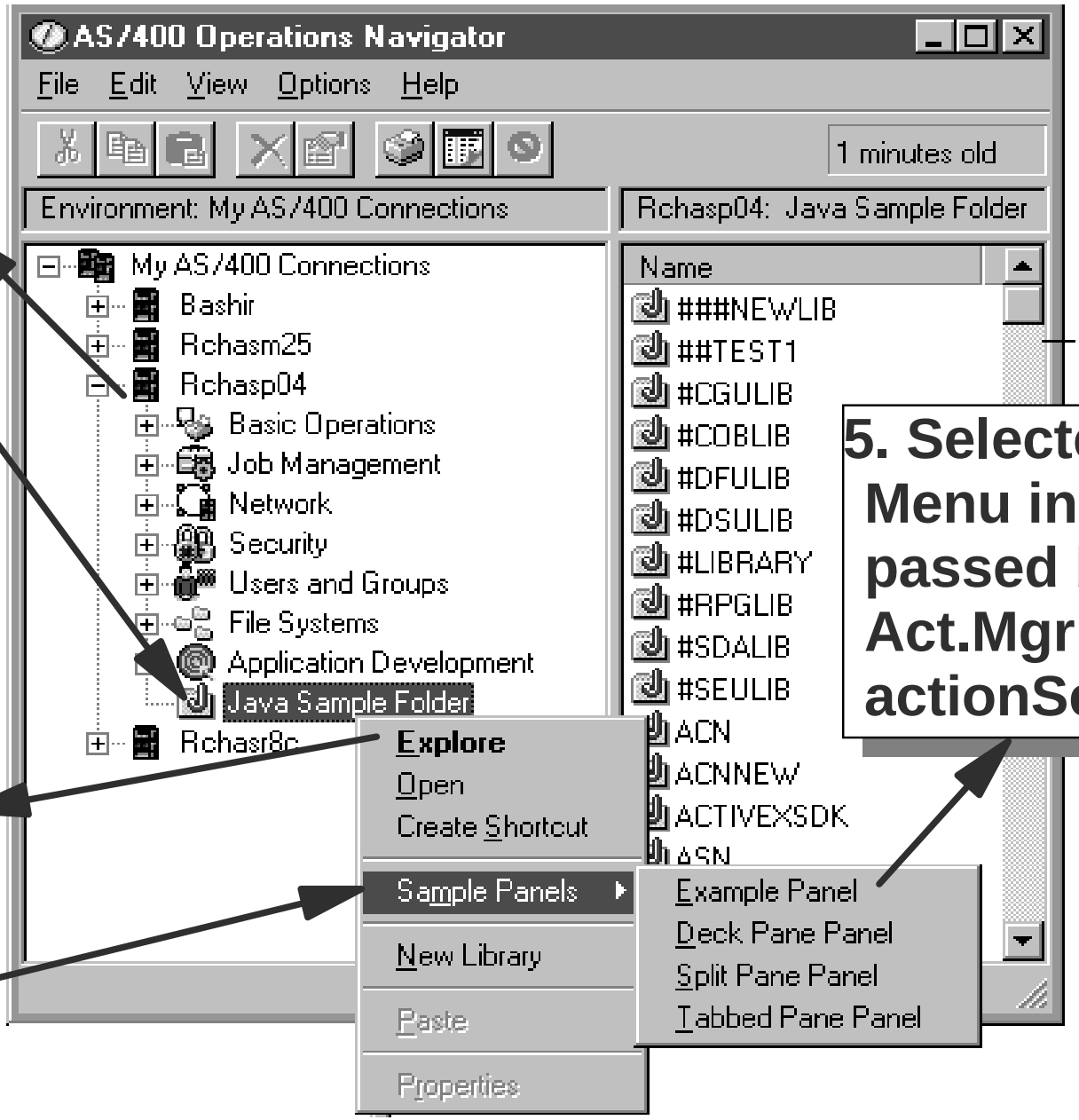
ActionsManager Diagram

1. Registry consulted for Plug-ins

2. Icon & text from MRI.DLL

3. On right click, Act.Mgr initialized, queryActions invoked

4. Menu items added

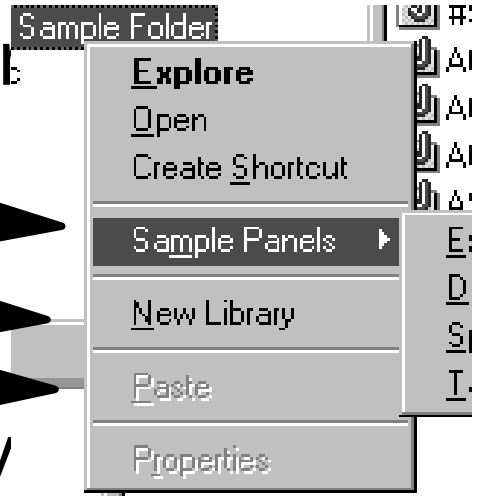


5. Selected Menu index passed back to Act.Mgr actionSelected

ActionsManager Example (con't)

queryActions (building context menu)

- get the object type
- handle context for drag-drop first
- check for different types
- check for action zones in context menu
  - CUSTOM
  - CREATION
  - STANDARD
- add ActionDescriptors to action array
- ***frequently called - be efficient!***

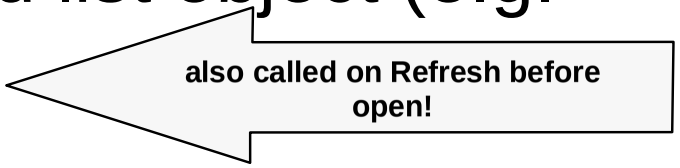


ListManager Interface

- Construct & manage a list of objects
- Required to populate OpNav component lists
  - initialize - identifies the container object to be enumerated
  - open - obtain and construct the objects (called on separate "data" thread - ***do long stuff here!***)
  - getStatus - returns the current status of the list
  - getErrorMessage - if getStatus is LIST_ERROR
  - getItemCount - should return the total number of object in the list
  - itemAt - returns ItemIdentifier of nth object

ListManager Interface (con't 2)

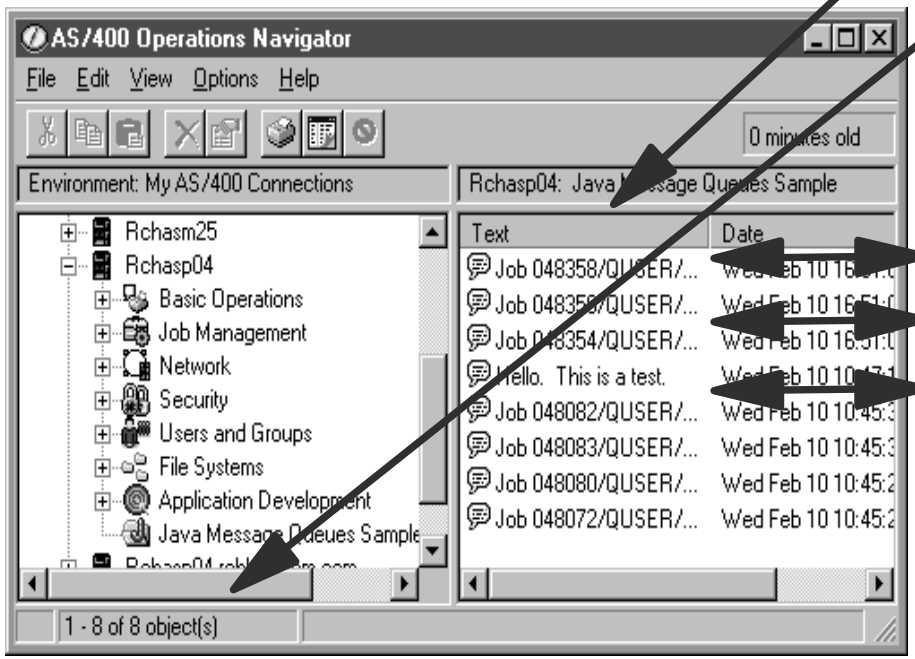
- getAttributes - attribute flags for nth object
- getColumnInfo - returns column headings
- getColumnData - returns data for column list
- getListObject - returns cached list object (e.g. for another interface)
- close - frees all resources
- prepareToExit - last chance to clean up.
- ~~getImageFile - returns image file name for nth object~~ (future expansion - currently unused)
- *getIconIndex - separate, non-interface method currently provides icon for nth object*



ListManager Example Diagram

1. On open ListManager initialized, passed object, open called

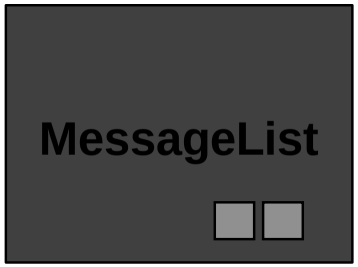
2. Open creates list. (Threaded) Sample creates a databean, which creates list class MessageList

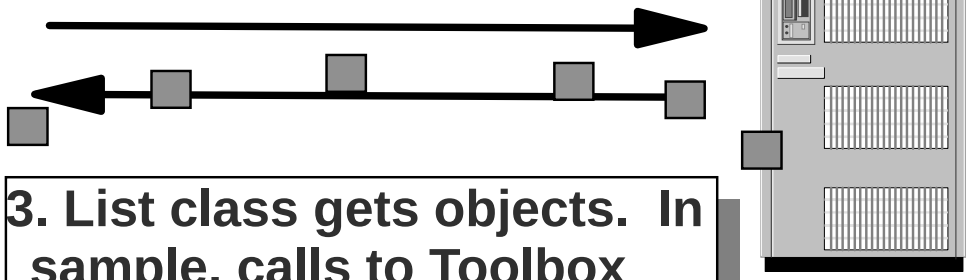


4. getColumnInfo (after thread return)

5. getItemCount

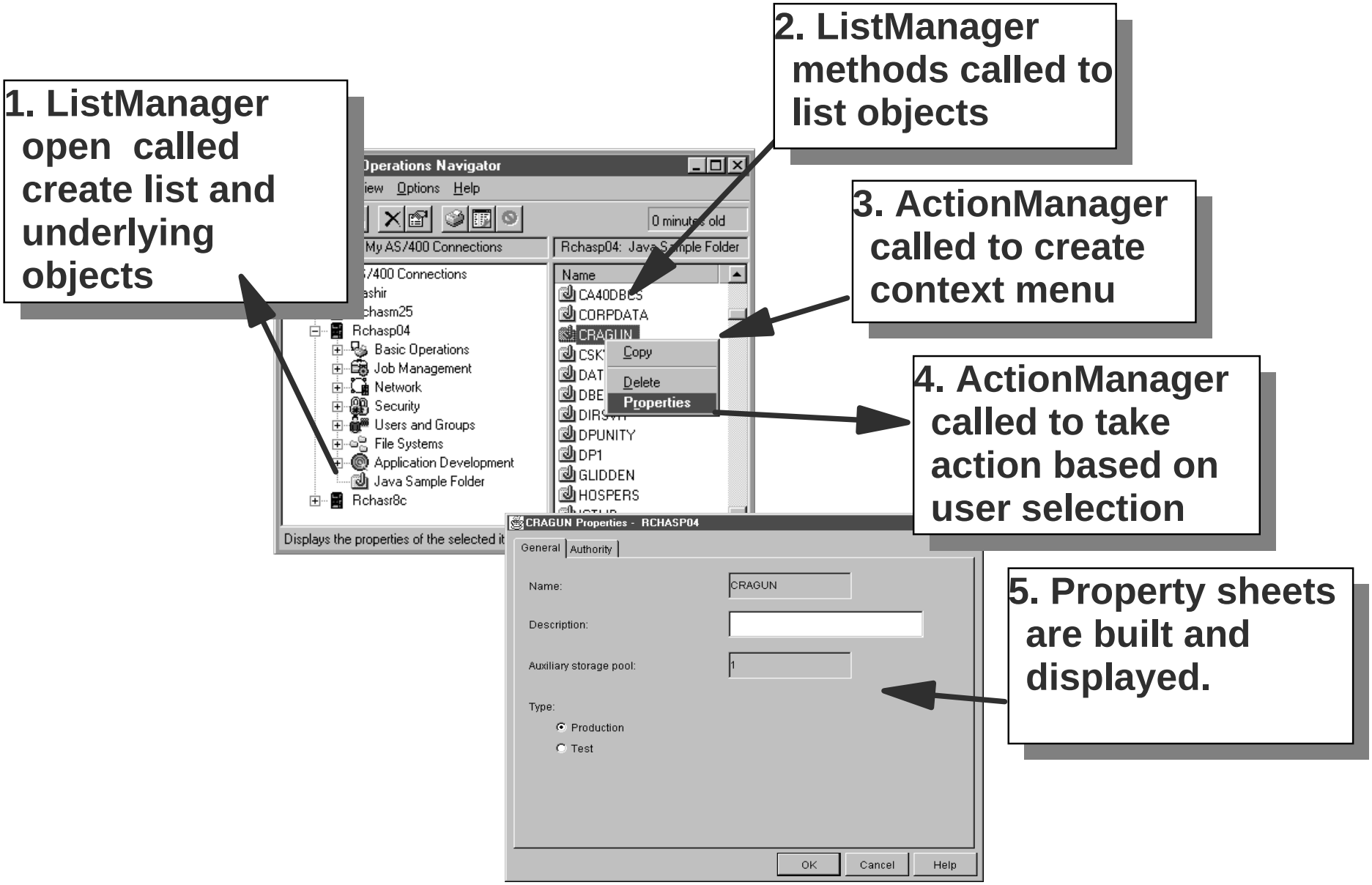
6. Repeated calls to fill list: itemAt, getAttributes, *getIconIndex*, getColumnData





3. List class gets objects. In sample, calls to Toolbox MessageQueue object to get message information

PropertyPage Example Diagram



Toolbars

- You provide additional MRI info
  - toolbar bitmap
  - toolbar information
- queryActions called for each button
- pressing button generates action
- actionSelected called with identifier of action
  - can be same action as menu action

Toolbar Example Diagram: How does the toolbar work?

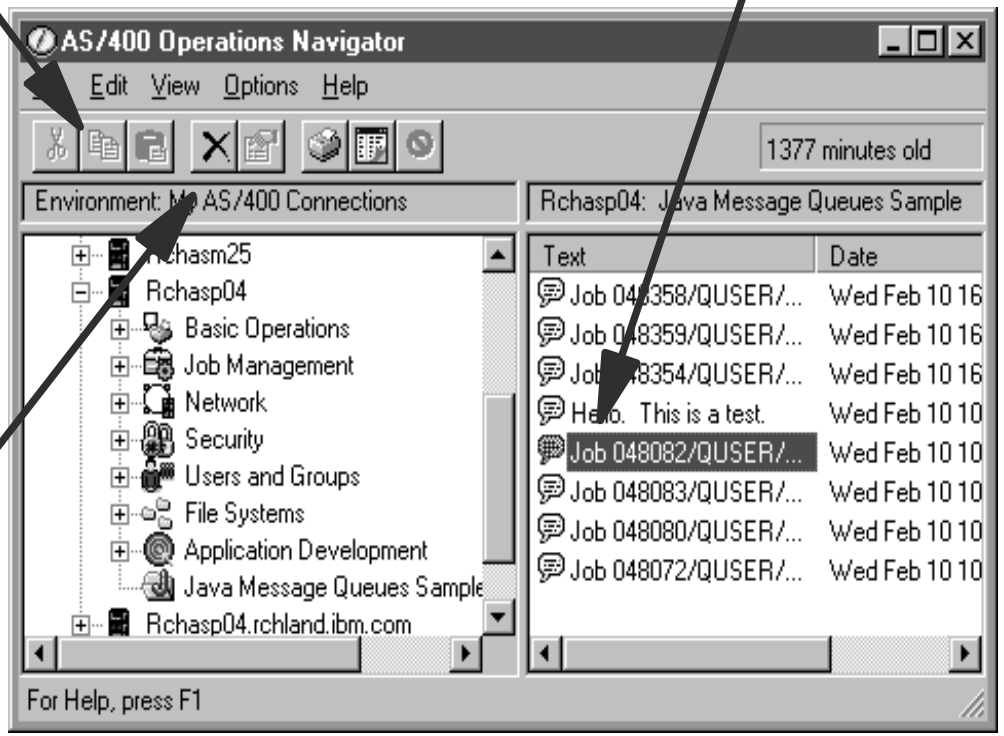
1. User selects object.

2. PanelManager processes each button.

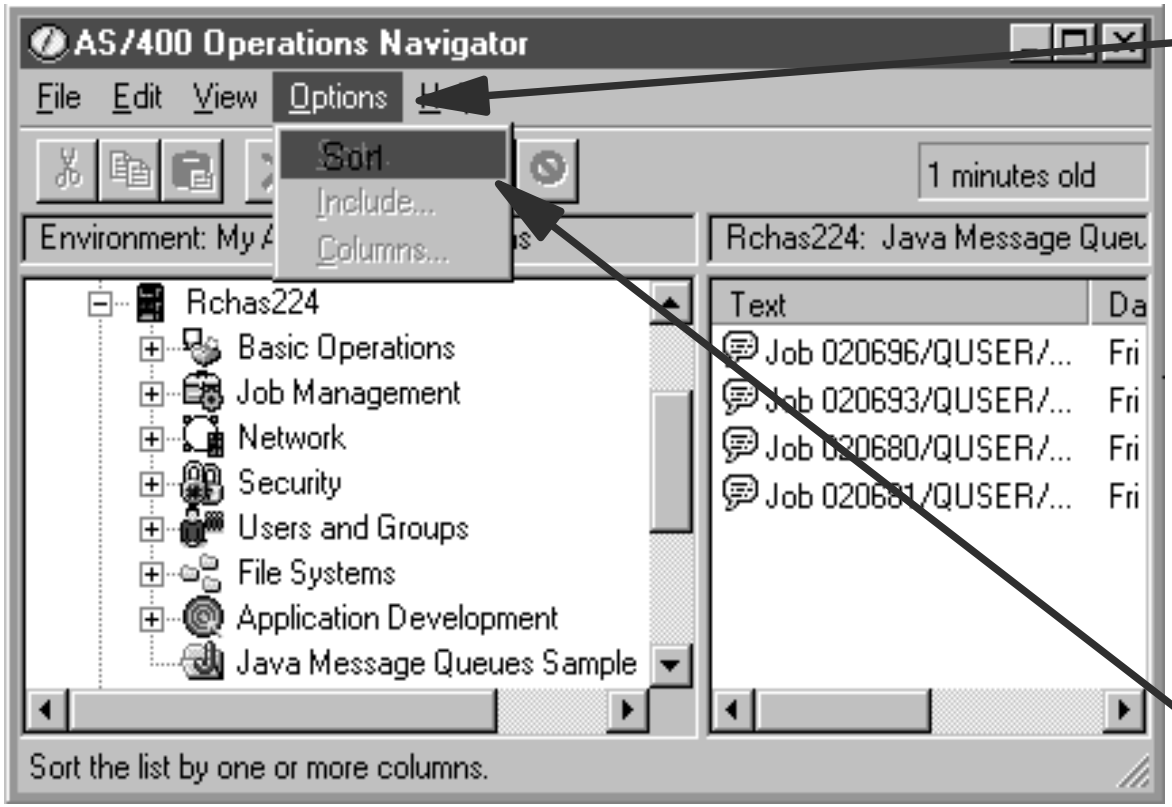
3. queryActions called

4. PanelManager checks if button verb is found in ActionDescriptor

5. PanelManager highlights or greys out accordingly



Options Example Diagram: How does the Option menu work?



1. User selects menu
2. queryActions called with OPTIONS_ACTIONS
3. ActionDescriptor created with SORT verb
4. PanelManager checks for SORT verb in ActionDescriptor
5. PanelManager highlights or greys menu accordingly

Options Menu Additions

OPTION_ACTIONS flag
SORT / INCLUDE / COLUMNS menu items

```
if (objectType.equals("MyObject"))
{
   if  (flags & OPTION_ACTIONS) == OPTION_ACTIONS   )
   {
      actions = new ActionDescriptor[1];  // Create an array to contain the returned
                                             descriptor
      ActionDescriptor d;                 // Create the descriptor for "MyOption"
      d = new ActionDescriptor(4);
      d.setText(m_loader.getString("menu.text.myoption")); // ignored
      d.setHelpText(m_loader.getString("menu.help.nyoption")); // ignored
      d.setVerb("INCLUDE");
            d.setVerb("SORT");
            d.setVerb("COLUMNS");
      actions[0] = d;   // Set it in the array
   }
}
```

you set action id

verb used to match toolbar verbs

DropTargetManager Interface

Handles drag and drop processing

- initialize - identifies target of drag/drop operation
- dragEnter - indicates whether a drop can be accepted and the action that will take place on the drop
- dragOver - called while the user continues over drop target (usually the same as dragEnter)
- dragExit - user left the target or cancelled
- drop - user indicates drop. Action is taken.

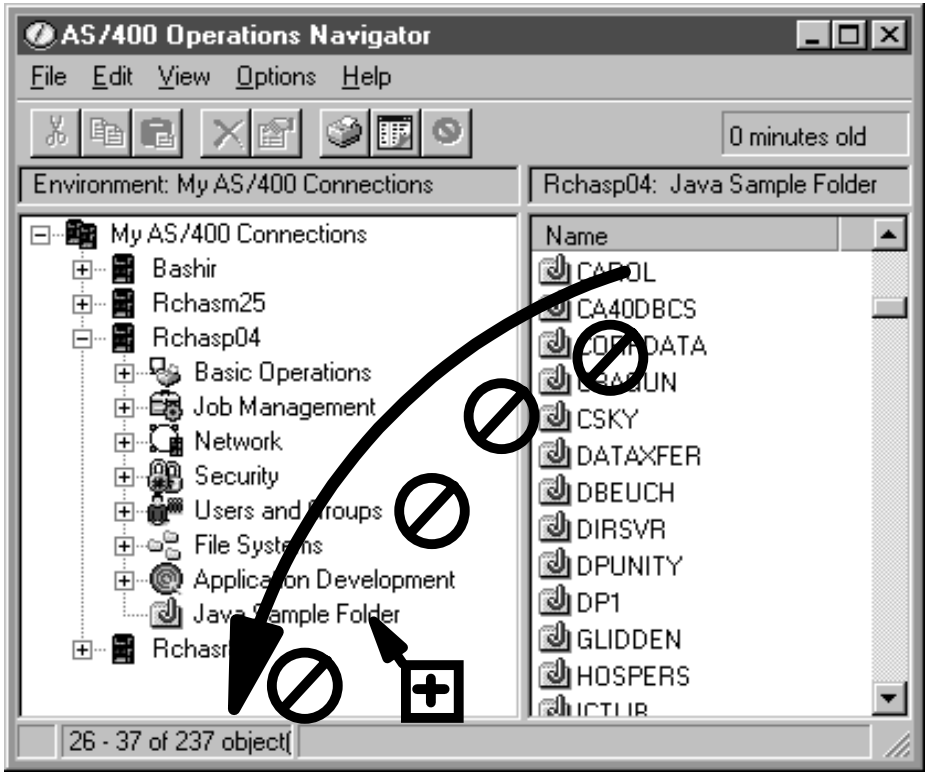
DropTargetManager Diagram

1. Start drag detected

2. OpNav calls Drag/Drop handlers for each target

3. Each handler determines if drop allowed

4. Java DropTargetMgr. called



5. dragEnter checks DropSource

If approved displays drop symbol

If not approved displays no drop symbol

6. dragOver rechecks and redisplays

7. drop handles drop action

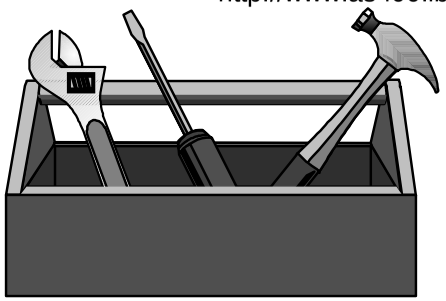
8. dragExit resets drop symbol to no drop

Programmatic Identifier (ProgID)

- Indentifies your Plug-in uniquely
- Form: *vendor* **.** *component*
- Example: Lotus.Domino, IBM.Sample
- Used in registry file and in the directory structure of the client

Development and Tools

Development and Tools



- **Portions of Java development are made easier with specialized mark-up languages**
  - Graphical Toolbox for creating GUIs for Java using XML
  - Program Call Markup Language (PCML) for AS/400 data access
- **New development environment automates using specialized mark-up languages**
- **Graphical Toolbox and PCML are packaged with the AS/400 Toolbox for Java**

XML and PDML Overview

What is XML?

- XML (Extendible Markup Language) allows you to define your own tagged-based language.

What is PDML?

- PDML (Panel Definition Markup Language) is a set of tags that define a panel, what is on the panel, how it is layed out, etc.

More about PDML

PDML is a language that we developed using XML. It's similar in structure to HTML and SGML.

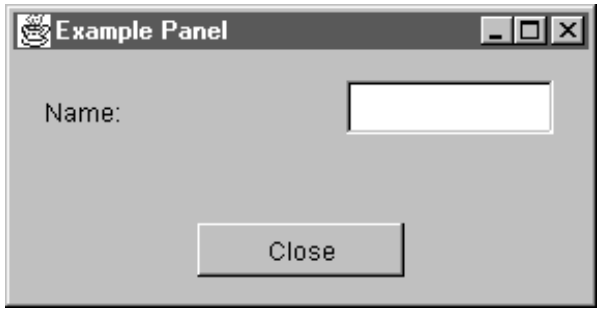
Here are some examples of PDML tags:

- <panel> -- defines a panel
- <title> -- specifies the title of the panel or field
- <size> -- specifies the size of the panel or field
- <label> -- defines a label on the panel (static text field)
- <location> -- specifies the location of the field on the panel
- <button> -- defines a button on the panel
- <textfield> -- defines a textfield on the panel
- and many more...

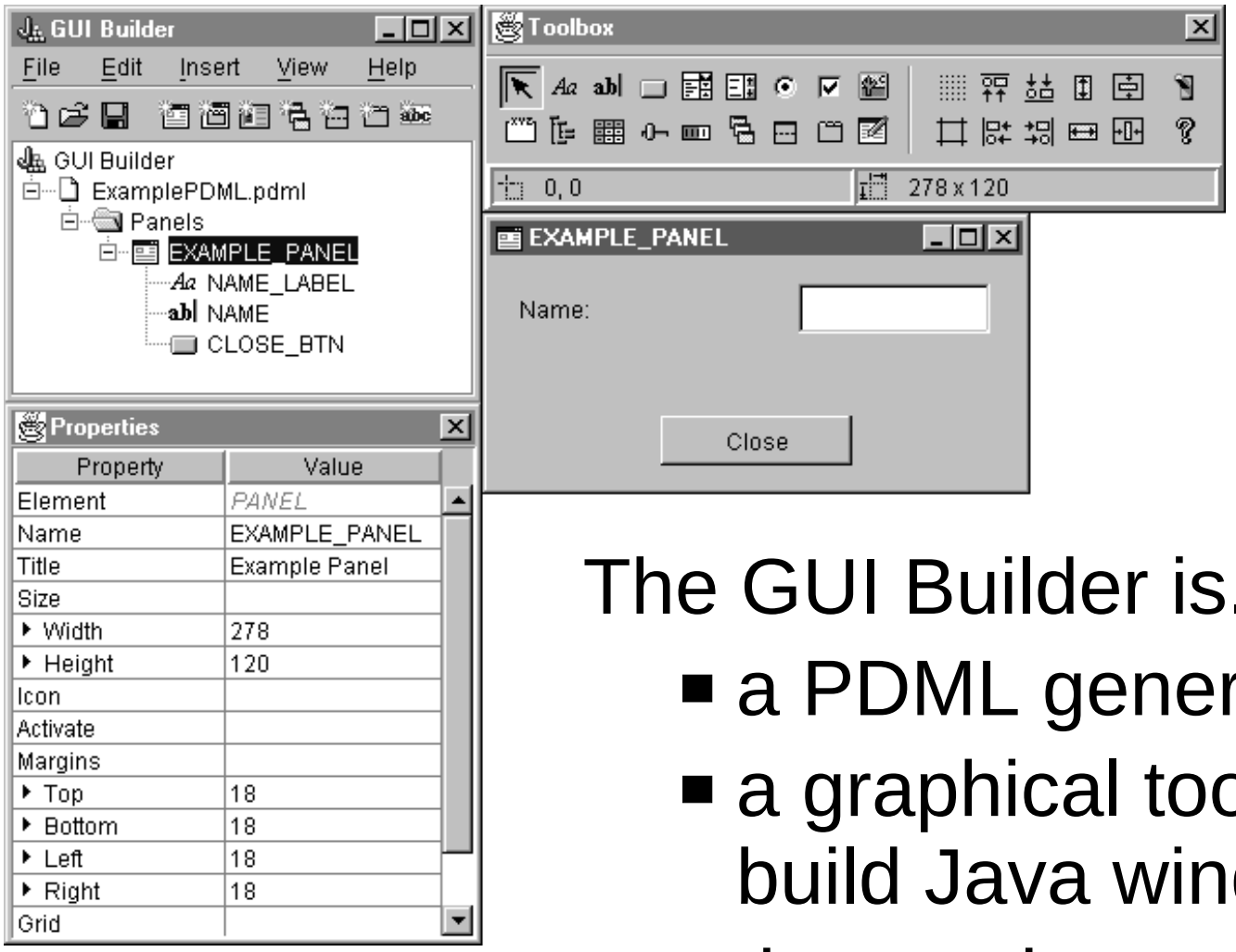
A PDML Example

Here's what the PDML looks like for the simple panel shown below:

```
<PDML version="1.0" source="JAVA" basescreensize="1024x768">

        <PANEL name="EXAMPLE_PANEL">
                <TITLE>EXAMPLE_PANEL</TITLE>
                <SIZE>278,120</SIZE>
                <LABEL name="NAME_LABEL" disabled="no">
                        <TITLE>NAME_LABEL</TITLE>
                        <LOCATION>15,20</LOCATION>
                        <SIZE>100,19</SIZE>
                </LABEL>
                <TEXTFIELD name="NAME" masked="no" editable="yes"
                                disabled="no">
                        <TITLE>NAME</TITLE>
                        <LOCATION>161,14</LOCATION>
                        <SIZE>100,26</SIZE>
                </TEXTFIELD>
                <BUTTON name="CLOSE_BTN" disabled="no">
                        <TITLE>CLOSE_BTN</TITLE>
                        <LOCATION>89,83</LOCATION>
                        <SIZE>100,26</SIZE>
                </BUTTON>
        </PANEL>

</PDML>
```



Simpler Java Dialogs with GUIBuilder



The GUI Builder is...

- a PDML generator
- a graphical tool used to build Java windows
- the starting point for all Java development!!

What can I do with the GUI Builder?

- Using the GUI Builder, you can:
  - Create GUIs
  - Manage the translatable text
  - Generate Help text
  - Generate Java Beans for data management
- Avoids hand-editing of PDML files, which may introduce parsing errors

Creating Java Windows

The GUI Builder allows you to create different types of Java windows:

- Panels -- simple dialogs
- Property Sheets -- Notebook pages with tabs at the top
- Wizards -- multiple pages displayed sequentially
- Tabbed Panes -- multiple pages with tabs at the top, bottom, right or left
- Split Panes -- multiple pages displayed adjacent horizontally or vertically
- Deck Panes -- multiple pages layed on top of each other
- Menus -- menu bars containing menu items
- Toolbars -- toolbars with menu items that can link to existing menu items in the menu bar
- Context Menus -- context menu definitions that can be used to pop-up context menus for components

Creating Java Windows

Steps to create Java Windows in the GUI Builder:

- Create a "Panel"
- Add the Fields to the Panels using the Toolbox
- If creating a multi-page window:
  - Create the additional "Panels" for each page in the multi-page window.
  - Create the Property Sheet, Tabbed Pane, Split Pane, or Deck Pane.
  - Add the panels to the multi-page window.
- Enter properties for controls
- Click on the Preview button to verify it looks correct.

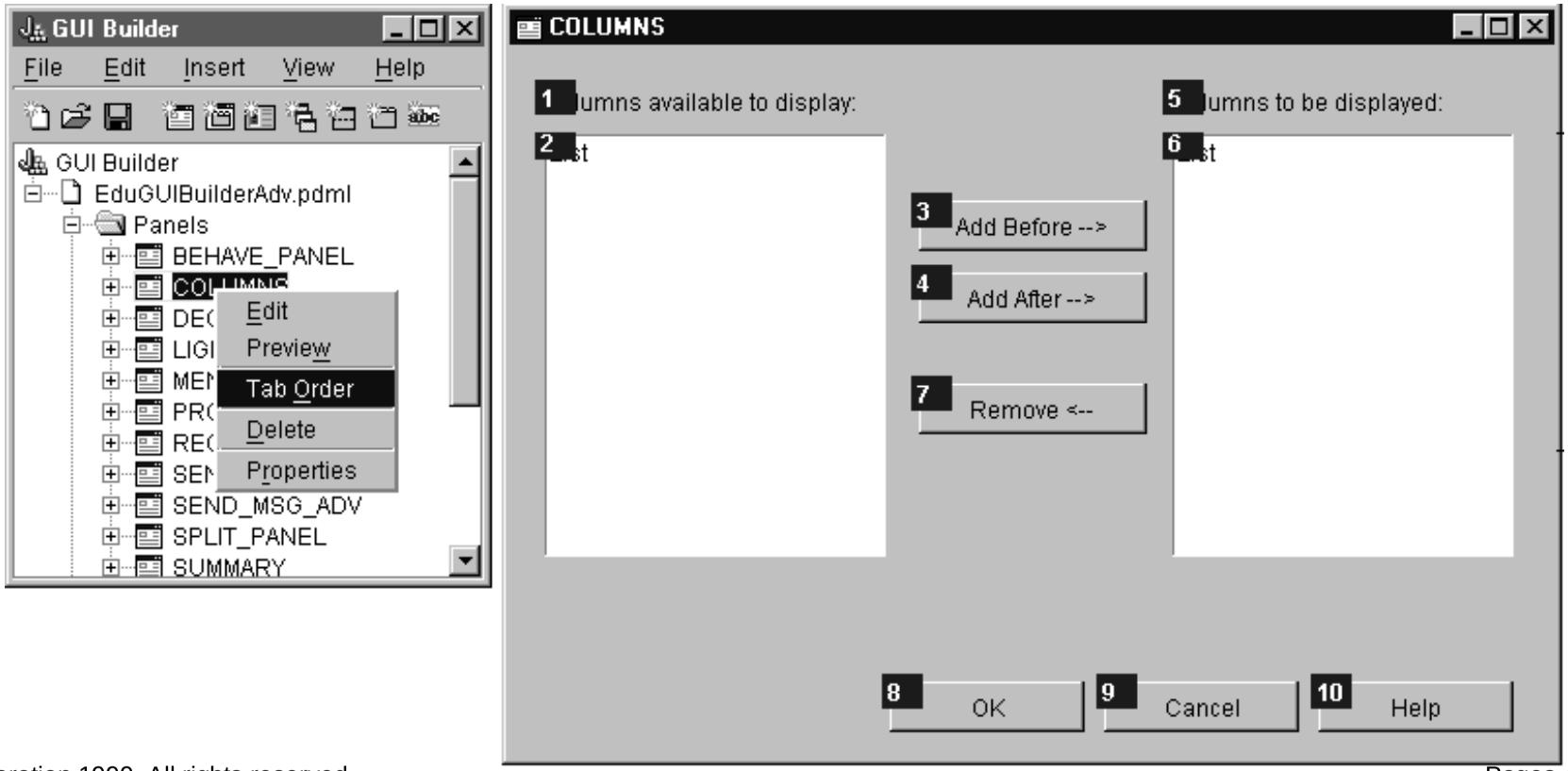
Creating Java Windows

Field types that you can add to your panels:

- Label
- Edit box
- Button
- Drop down list
- List box
- Radio button
- Check box
- Image
- Group box
- Tree
- Table
- Slider
- Progress bar
- Custom
- Buttongroup for radio buttons
- Spin buttons (future)

Creating Java Windows

Other behaviors: You can use the GUI Builder to set up the tabbing order through your windows. Tabbing order is important for accessibility.



Managing translatable text

- The GUI Builder provides the String Table function in order to manage translatable text.

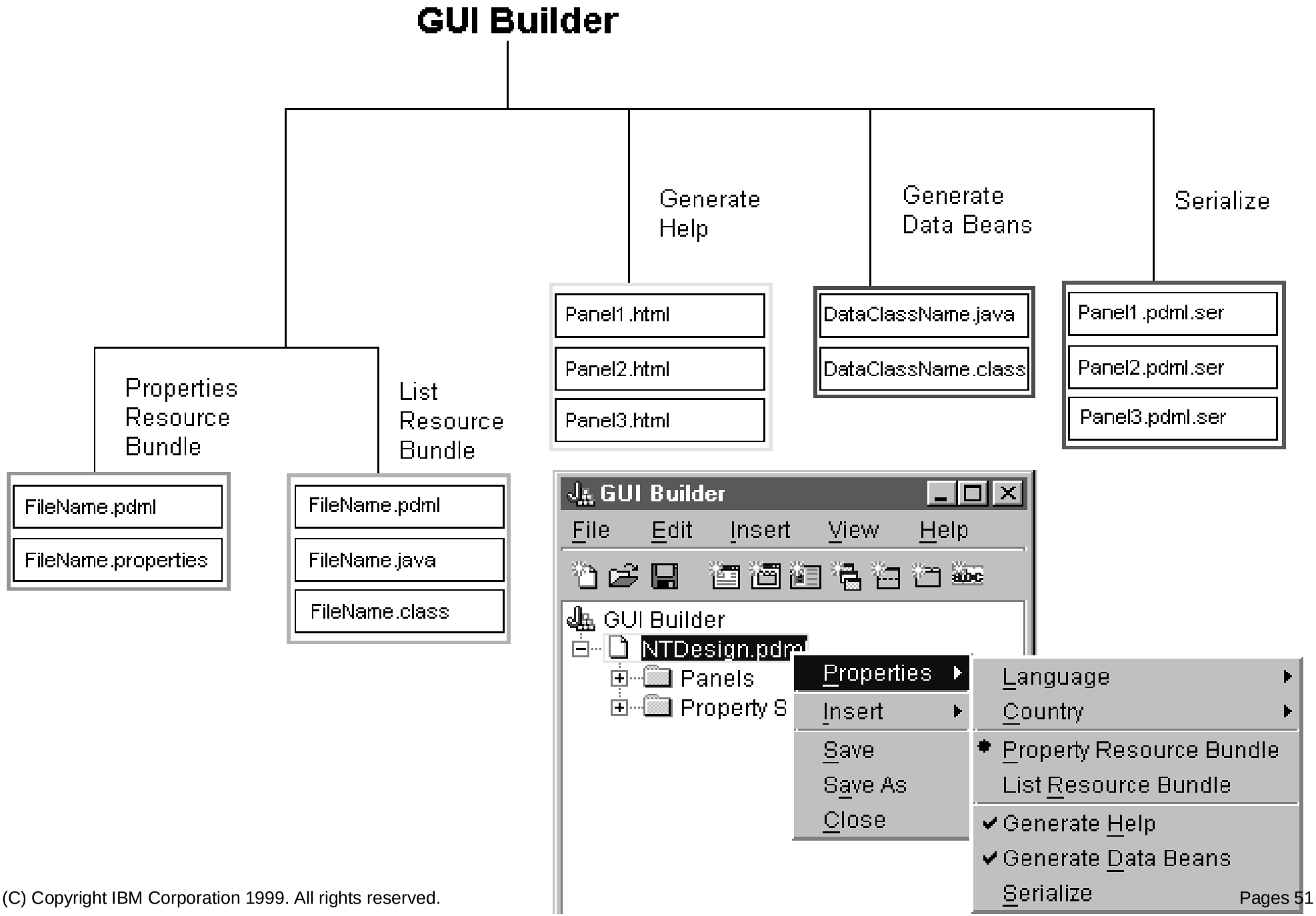
- Use the String table for:
  - Messages
  - Menu text and status bar help
  - Additional status bar information (e.g. User Hlucke created)

Managing translatable text

What about the text on the panels?

- The field text on each panel is translatable.
- The field text does not need to be added to the string table.
- The field text is stored in the same file as the string table information. During translation, it will also get translated.

Files generated by GUI Builder



Files generated by GUI Builder

- Properties Resource Bundle
  - By default, it uses the Properties Resource Bundle which will create two files for you:
    - **FileName.pdml** -- holds the panel definition information, such as the type of fields, their location, etc.
    - **FileName.properties** -- holds all of the translatable text
- List Resource Bundle
  - Used to enhance performance of the application.
  - Compiles the panel definitions and translatable text, so performance is better at run-time.
    - However, it will slow down performance in the GUI Builder when previewing panels.
  - This bundle will create three files for you:
    - **FileName.pdml** -- holds the panel definition information, such as the type of fields, their location, etc.
    - **FileName.java** -- holds all of the translatable text
    - **FileName.class** -- compiled version of FileName.java

Files generated by GUI Builder

Generating Help

- Generates help skeletons for each window in the PDML file. This help skeleton will appear when the user clicks on the Help button or when they press F1.
- Does not overwrite any existing Help in the Help skeleton.
- Creates these files for you:
  - **PanelName.html** (for each panel in the PDML file)

Files generated by GUI Builder

Generating Data Beans

- Generates data beans (get and set methods) for fields in the PDML.
- Makes it easy for you to get started on the Java code for the panels.
- Creates these files for you:
  - **DataClassName.java** (for each data class specified in the PDML file) -- file with get and set methods for each attribute in the data class
  - **DataClassName.class** (for each data class specified in the PDML file) -- compiled version of the corresponding .java file
- In the future, event handlers will also be generated

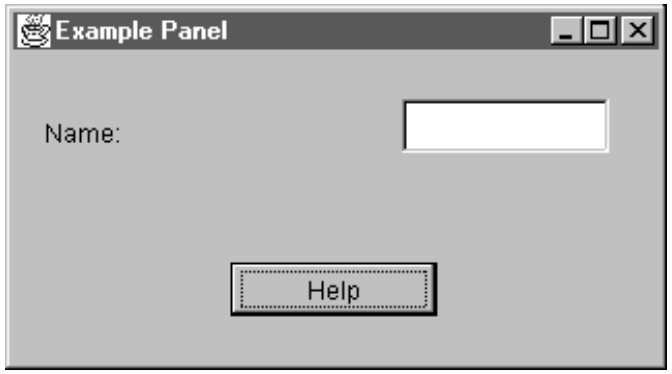
Files generated by GUI Builder

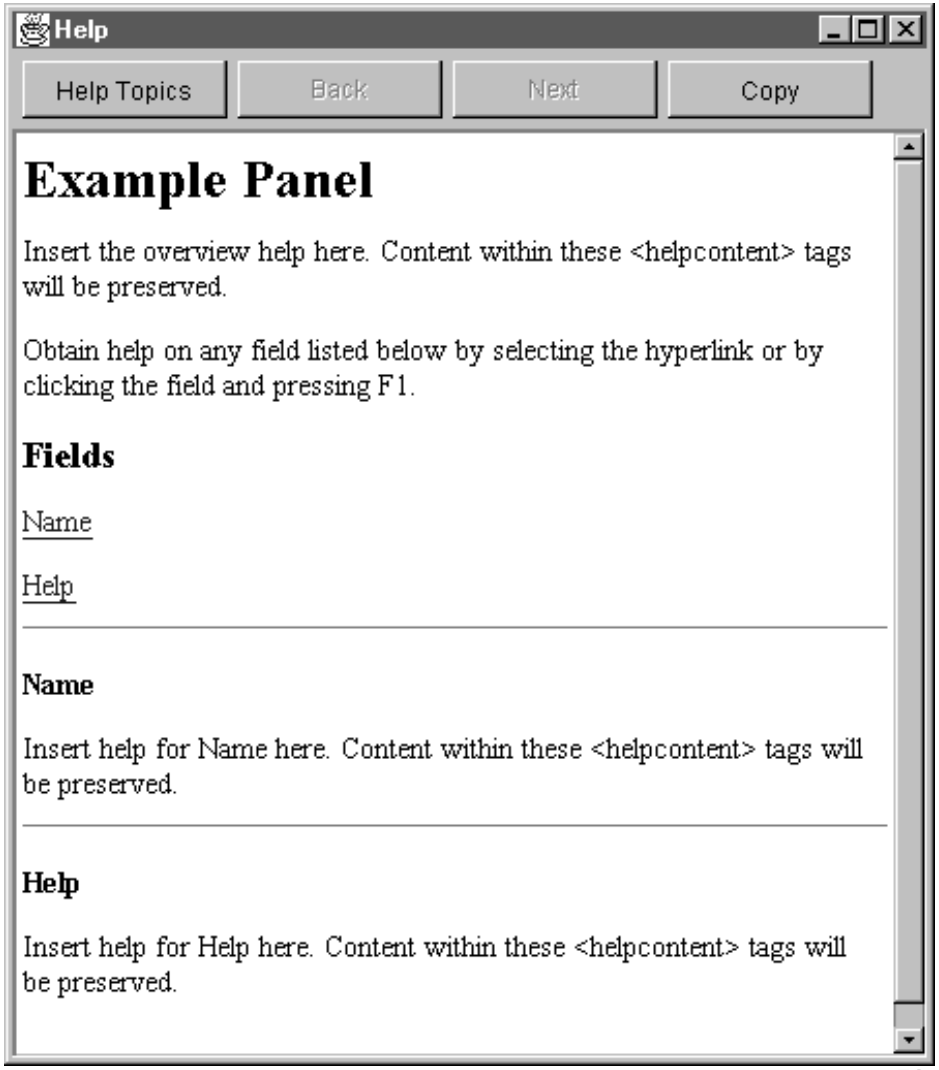
Serialization

- Compresses the information for each window in the PDML file.
- Enhances performance of your Java panels.
- Creates these files for you:
  - **PanelName.pdml.ser** (for each panel in the PDML file)

Help Generation

- Creates help skeletons for each panel in the PDML file.
- These Help skeletons will be used by the help writer (not by the developer) to build the help for each panel.
- The help file cannot be tested in the GUI Builder. You must load the file in a Java application for the help to appear.





Help Generation

Which fields get put into the skeleton?

- radio buttons
- check boxes
- group boxes
- buttons
- labels that end with colons

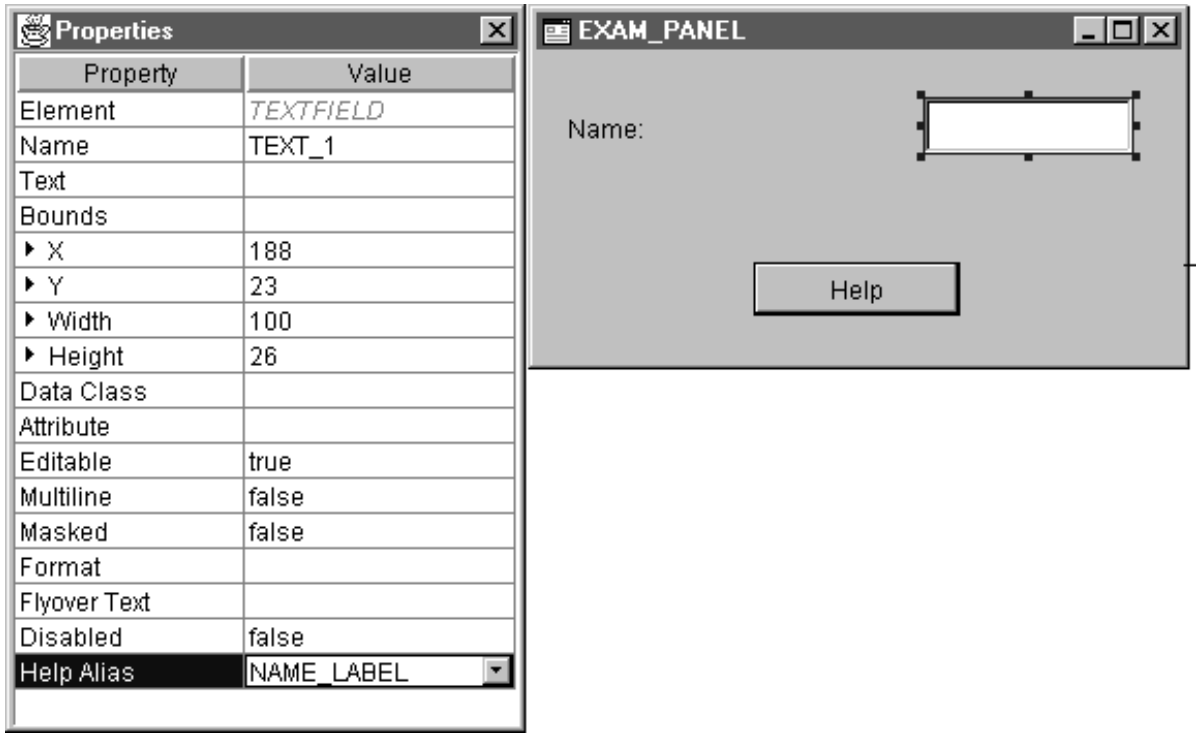
But what about lists, tables, combo boxes, edit boxes, etc.?

- If you want F1 to bring up Help on these fields, you need to set up a **Help Alias** for each field in the GUI Builder.

Help Generation

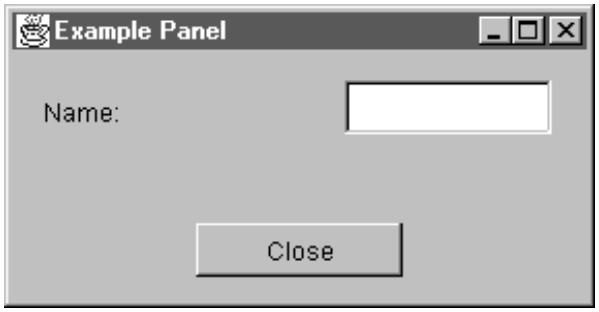
Help Alias

- points to another field which already has help, such as a label that ends with a colon.
- Specify the Help alias in the Properties



Data Bean Generation

Creates data bean skeletons (get and set methods) for each data class and attribute specified in the PDML file.

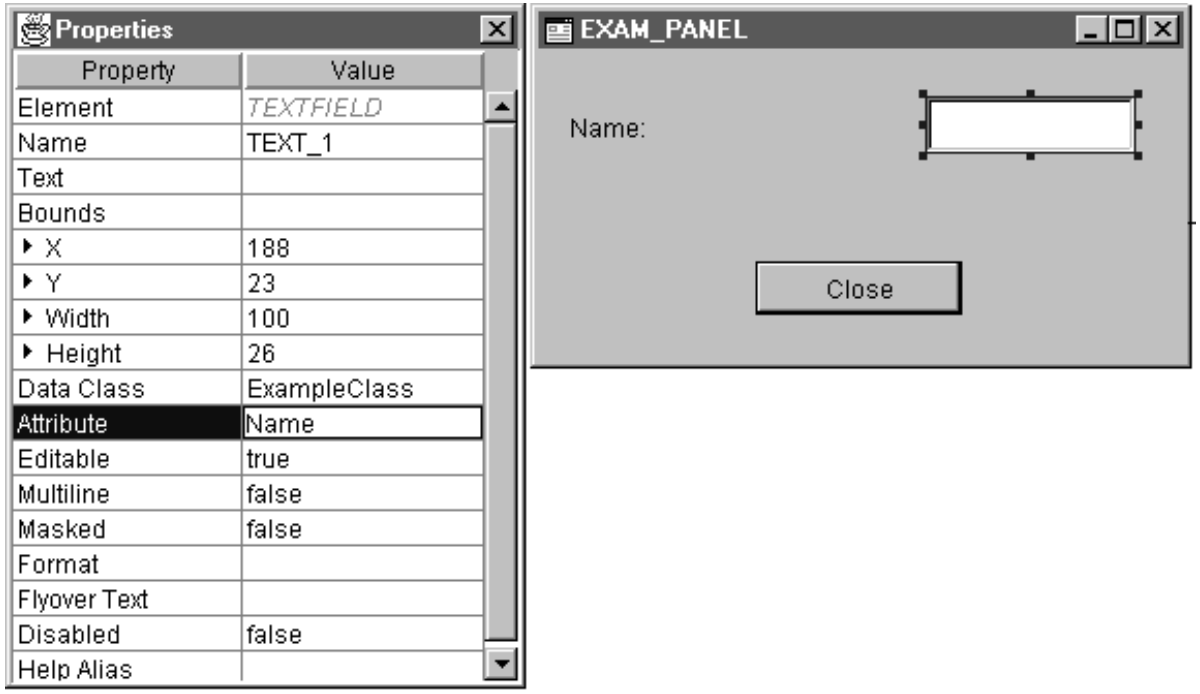



```
import com.ibm.as400.ui.framework.java.*;

public class ExampleClass extends Object implements DataBean
{
    private String m_sName;

   public String getName()
    {       return m_sName;     }

    public void setName(String s)
    {       m_sName = s;    }

    public Capabilities getCapabilities()
    {       return null;    }

    public void verifyChanges()
    {   }

    public void save()
    {   }

    public void load()
    {       m_sName = "";    }
}
```

Data Bean Generation

Data Class and Attributes

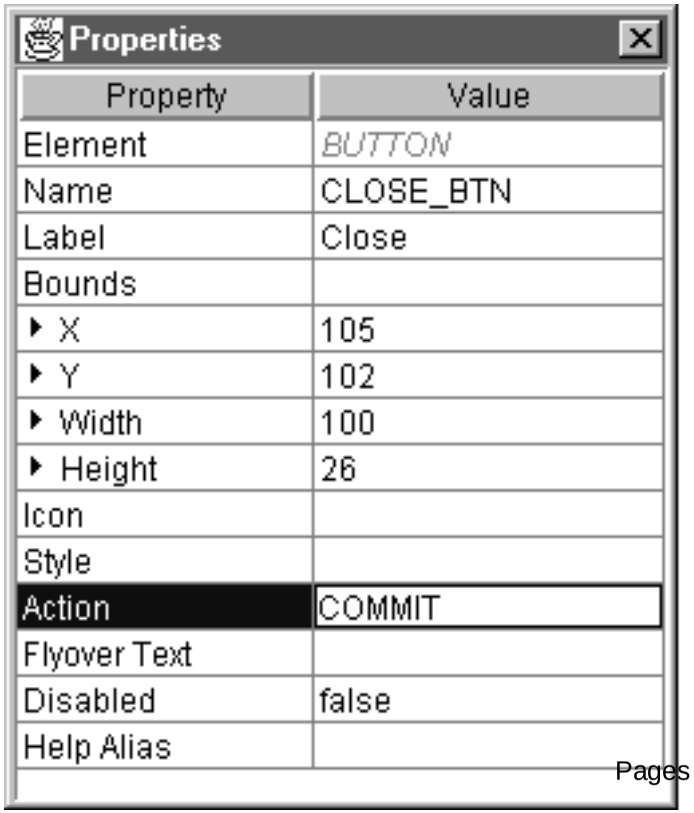
- used to send and receive information to and from individual fields on the panels.
- Specify the Data Class and Attribute in the Properties



Button Handling

Button Actions

- Specify each button's Action handler in the Properties (by specifying the class name)
- Does not generate the button handler (In V4R5, it will generate a button handler skeleton)
- Three Button Action's are available without having to write a button handler (must specify in Properties):
  - COMMIT -- For OK buttons
  - CANCEL -- For Cancel buttons
  - HELP -- For Help buttons



Development

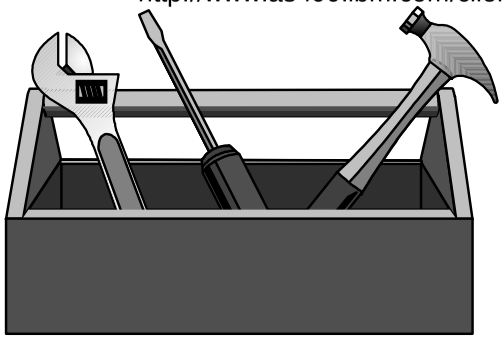
- GUI (what does it look like)
- Functions (what does it do)
- MRI icons / dialogs / strings
- AS/400 access
  - classes (toolbox)
  - AS/400 system APIs
  - JDBC
  - CL commands
  - PCML
- Parts packaging
- Delivery vehicle
- Install process

PCML -
Simpifying AS/400 Data Access

Getting data to/from the AS/400

- Java Toolbox objects
- CL commands
- JDBC
- AS/400 API's
- PCML

Cool objects in the AS/400 Toolbox for Java



- Users and groups
- User spaces
- Data queues
- Messages and message queues
- Digital certificates
- System values
- System status
- Object authority
- Jobs and job attributes
- Data areas

etc.

Introduction to PCML

- **Program Call Markup Language (PCML) was created to simplify the process of calling AS/400 programs from Java Programs**
- **PCML is implemented as a package of java classes**
  - com.ibm.as400.data (data400.jar)
- **Built on top of the AS/400 Toolbox for Java access package**
  - com.ibm.as400.access (jt400.zip/jt400.jar)
- **Both shipped as part of the AS/400 Toolbox for Java**

What is PCML?

- Program Call Markup Language
  - An XML-based language to define AS/400 program interfaces
  - PCML describes an AS/400 program interface
    - Used to drive automatic data translation

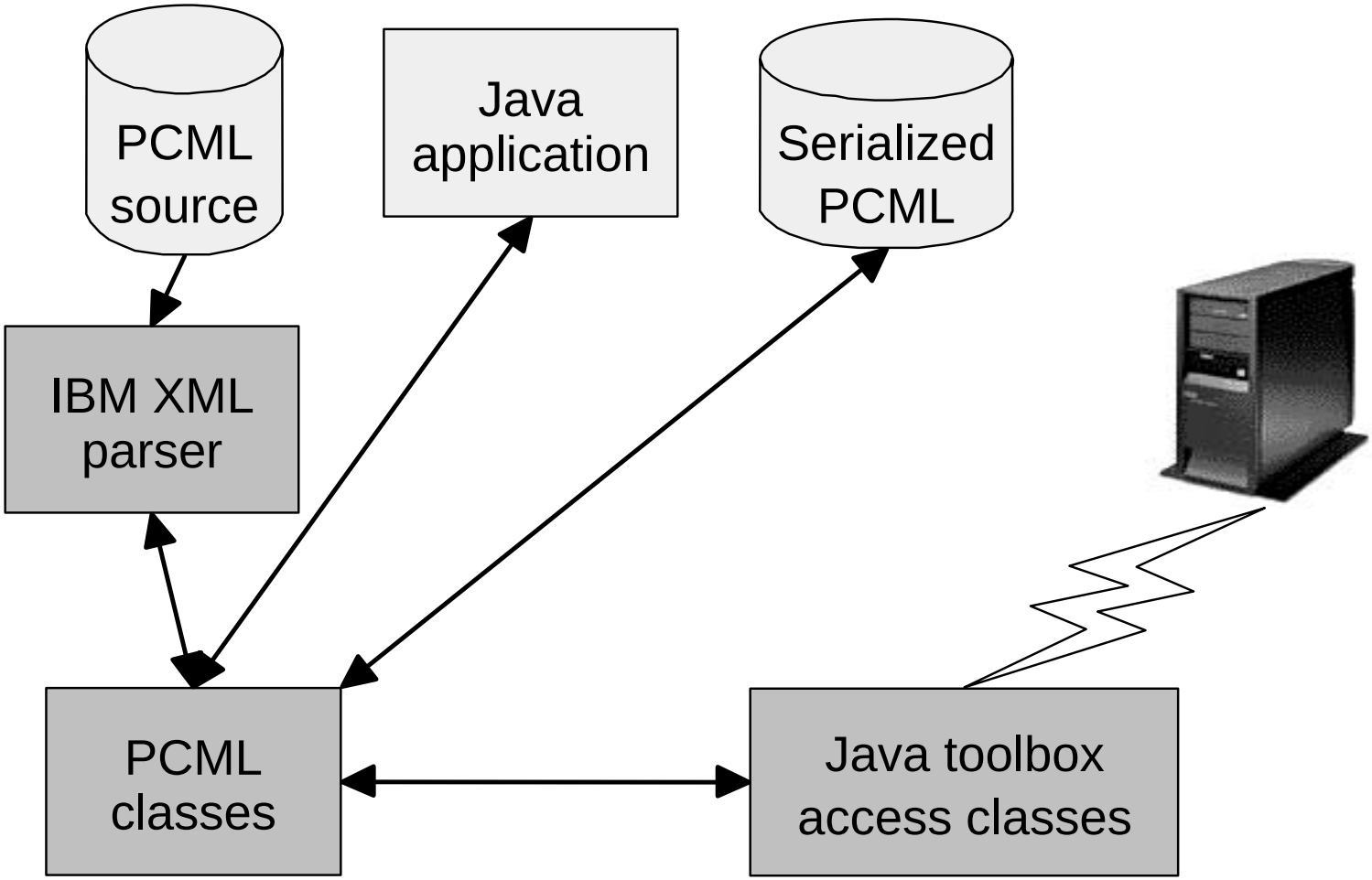
- com.ibm.as400.data.ProgramCallDocument
  - A Java class representing the interfaces described in a PCML source file
  - Performs data conversion and program calls to the AS/400

Simplifying AS/400 program calls

Simplifies Java programs by handling complex relationships in AS/400 data

- Varying length character strings
- Varying length structures
- Varying sized arrays of fields and structures
- Nested arrays
- Varying locations of data -- offsets and displacements
- Strings with run-time CCSID tagging
- Parameters or fields within a structure that exist only at specific OS release levels

Structure using PCML



Simple example - PCML source

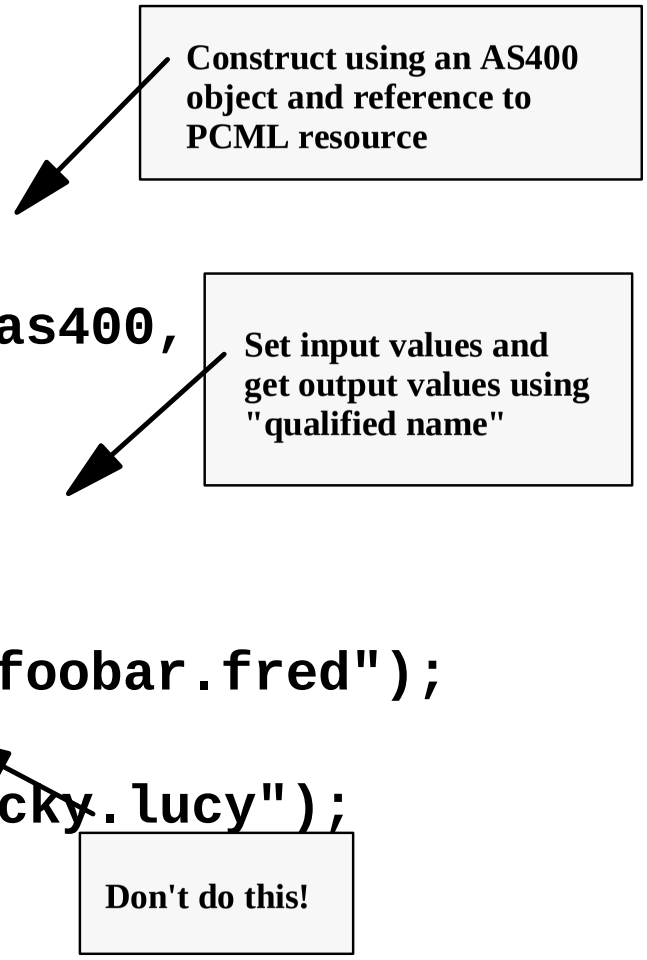
```
<pcml version="1.0">
   <program name="foobar"

path="/QSYS.lib/JOE.lib/RABOOF.pgm">
      (1) <data name="fred"  type="char"
lengt(2)="1" />
      (3) <data name="ethel" type="int"
length="4" />
      <struct name="lucyAndRicky">
         <data name="lucy" type="char"
length="1" />
         <data name="ricky" type="int"
len
</pcml>
```

name= does not have to match the name of the actual program

<program> contains one tag for each parameter

Simple example - Java code

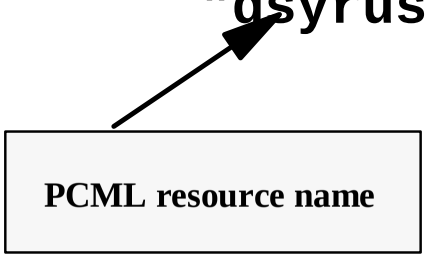
```
AS400 as400 = new AS400();
try {
   ProgramCallDocument pcml =
       new ProgramCallDocument(as400,
                   "example");

   pcml.callProgram("foobar");

   Object val = pcml.getValue("foobar.fred");
   val =
pcml.getValue("foobar.lucyAndRicky.lucy");
}
catch (PcmlException pe) {}
```



The ProgramCallDocument class

- PCML is implemented by the com.ibm.as400.data package
- The public class **`ProgramCallDocument`** does all the work
- Constructed with an AS400 (toolbox) object, and the name of the PCML resource

```
pcml = new ProgramCallDocument(myAs400,
                "qsyrusri");
```



The `ProgramCallDocument` class *(continued)*

In the above example, "qsyrusri" is the name of a PCML resource to be located via the classpath.

- Constructor first looks for serialized version of resource
  - **`"qsyrusri.pcml.ser"`**
- If serialized version not found, source version must be found
  - **`"qsyrusri.pcml"`**
- Resource name can be package qualified on constructor
  - **`"com.ibm.mypkg.qsyrusri"`**

Names of <data> elements

- Data elements are accessed in Java by specifying the fully qualified name of the element.
- A qualified name consists of the qualified name of the element's parent, a period separator, followed by the element's **name=** attribute

For example:

```
pcml.getValue("qsyrusri.receiver.userName");
```

Comes from:

```
<program name="qsyrusri">
  <struct name="receiver">
    <data name="userName">
```

Data types

AS/400 type=	length=	precision=	Java object
char	0..65536 *data-name*		java.lang.String
int	2 4	15 (signed) 16 (unsigned) 31 (signed) 32 (unsigned)	java.lang.Short java.lang.Integer java.lang.Integer java.lang.Long
packed	1..31	0..*length*	java.math.BigDecimal
zoned	1..31	0..*length*	java.math.BigDecimal
float	2 4		java.lang.Float java.lang.Double
byte	0..65536 *data-name*		byte [ ]

Java object is input to

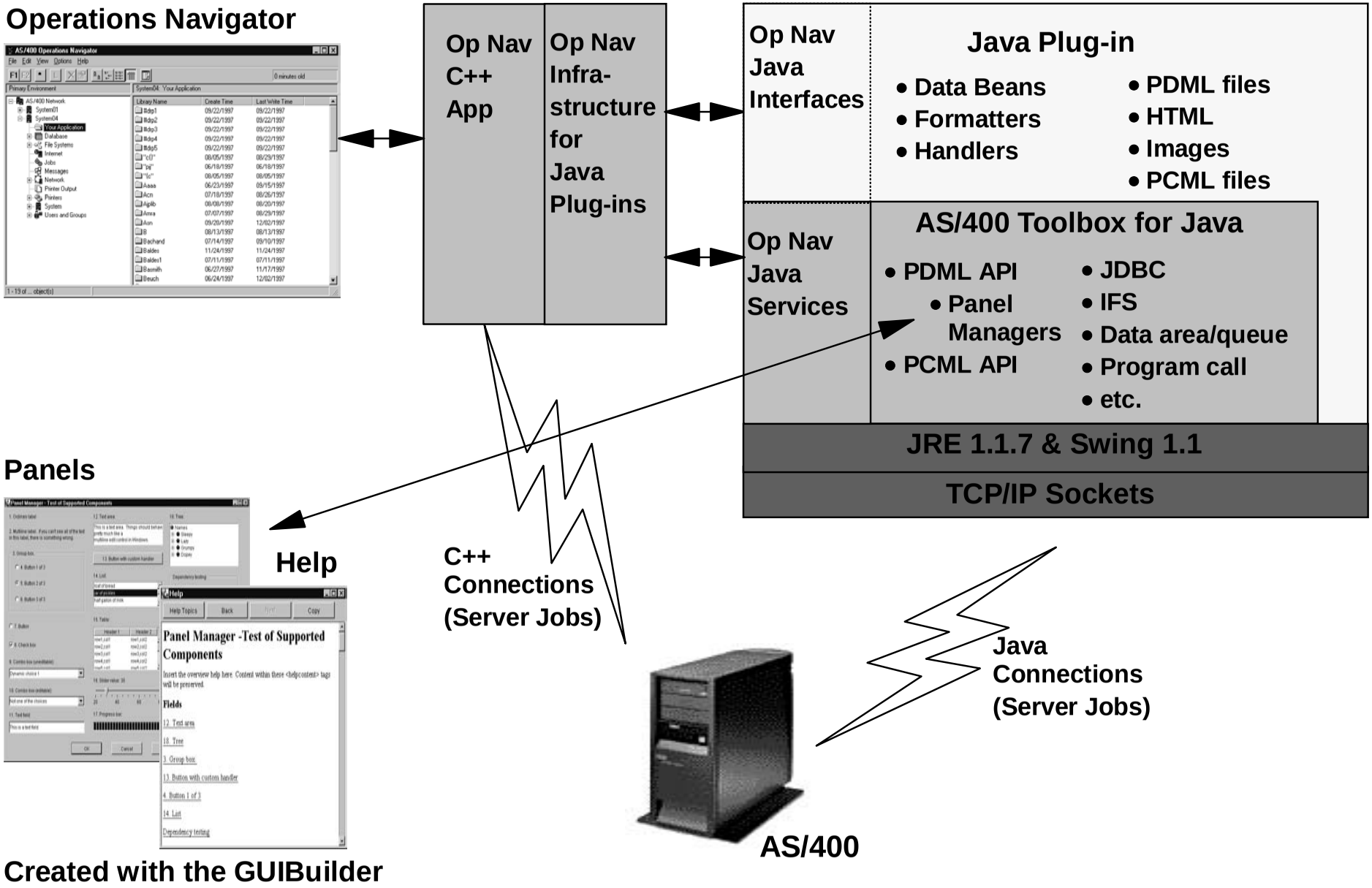
ProgramCallDocument.setValue()

and returned from

ProgramCallDocument.getValue()

Java Plug-in and Application Development

Operations Navigator with Java Plug-ins



Developing Java Plug-ins (for V4R4)

- Install Client Access Express - select AS/400 Toolbox for Java from Selective install menu
- Run CheckVersion to apply GA service pack
- Install Sun JDK 1.1.7
- Set up classpath:

```
   jdk1.1.7\lib\classes.zip;
   ClassPath=
    .;C:\java;C:\jdk1.1.7\lib\classes.zip;
    C:\Program Files\IBM\ClientAccess\jt400\lib\uitools.jar;
    C:\Program Files\IBM\Client Access\jre\lib\swingall.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;
    Path=...;C:\jdk1.1.7\bin;
```

Developing Java Applications (V4R4)

- Apply Toolbox GA PTF on AS/400
- Install Sun JDK 1.1.7
- Install Swing 1.1.0
- Run Toolbox Installer, specifying the OPNAV package
- Set up classpath:

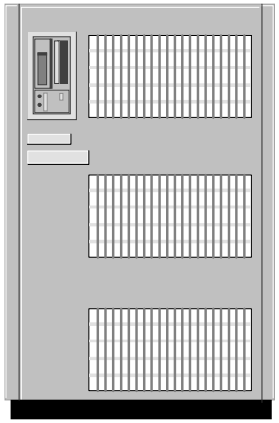
```
jdk1.1.7\lib\classes.zip;
swing-1.0.3\lib\swingall.jar;
jt400\lib\jui400.jar;
jt400\lib\uitools.jar;
jt400\lib\x4j400.jar;
jt400\lib\util400.jar;
jt400\lib\data400.jar;
jt400\lib\jt400.zip;
```

JAR Files

- **uitools.jar** Contains the GUI Builder and Resource Script Converter tools.
- **jui400.jar** Contains the runtime API for the Graphical Toolbox. Java programs use this API to display the panels constructed using the tools. May be redistributed.
- **data400.jar** Contains the runtime API for the Program Call Markup Language (PCML). Java programs use this API to call AS/400 programs whose parameters and return values are identified using PCML. May be redistributed.
- **util400.jar** Contains utility classes for formatting AS/400 data and handling AS/400 messages. May be redistributed.
- **x4j400.jar** Contains the XML parser used by the API classes to interpret PDML and PCML documents.
- **jt400.zip/jt400.jar** Contains the AS/400 Toolbox for Java jar .zip file

Installation of Plug-ins

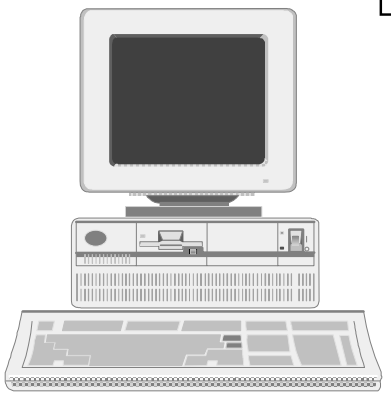
Installation



1. Product installs plug-in on server

2. Client uses Built-in selective Setup to install Plug-in

3. Plug-in copied and installed



4. Scanning activates Plug-in

Java Plug-in Installation

- Operations Navigator is shipped as part of Client Access Express at no additional charge.
- Delivered as part of a separate LPP
- Installed from an AS/400 using Client Access Selective Setup
- Must have it's own install option
- In all cases, installation of JRE 1.1.7, Swing 1.0.3 and the AS/400 Toolbox for Java is handled by Client Access Express

Install

- Built-in process in Client Access and OpNav

Client Access install image

  - Integral install
  - Registry entries and files in CA image

- 3rd Party install
  - Separate install to AS/400
  - Specialized .ini file
  - Load to client
  - Activation through "scanning"

Problem determination

Potential Problems

If the tools do not come up, the classpath is probably not specified correctly.

Fixes in the GA PTF:

- GUI Builder hangs on file save
- Language/country choices missing from tools
- Converter fails to clean up file viewtemp.properties
- Online help not displayed for certain countries
- GUI Builder error message always in English
- Generated names of UI controls always in English
- Missing blanks in Korean online help
- PCML attributes offsetfrom, minvrm/maxvrm don't work as documented
- Serialized PCML files named incorrectly when PCML document resource is package qualified

Problems in Production

- Graphical Toolbox writes error messages to a log file specified by the application
- PCML writes error messages to a log file specified by the application
- Operations Navigator writes error messages to the Client Access History Log & Detail Trace Log
- Operations Navigator plug-ins write error messages to a log file specified by the plug-in
- If problems occur, obtain all log files!

Problems during Development

- Graphical Toolbox writes error messages to the console (DOS window)
- Operations Navigator writes error messages to the Client Access History Log & Detail Trace Log
- Operations Navigator provides Java console window. To activate, set Windows registry entry:

```
   [HKEY_CURRENT_USER\Software\IBM\Client Access
Express\CurrentVersion\AS400 Operations Navigator\Trace]
   "JETRACE"="ON"
```

- If problems occur, obtain all customer files!
- PDML tracing available in com.ibm.as400.ui.framework.java
  - MessageLog.setTraceEnabled(true)
- PCML tracing available in com.ibm.as400.data
  - PcmlMessageLog.setTraceEnabled(true)

For more information.....

Samples for developers

The Client Access Express Toolkit contains samples for:

- Java (recommended)
- C++
- Visual Basic

★ Note, the Client Access Express Toolkit is an optionally installable component of Client Access Express.

For more information.....

- HTML help for the GUIBuilder and Resource Script Converter tools

- Info Center
  - For Plugin information via the Client Access Programming links
    - **http://publib.boulder.ibm.com/pubs/html/as400/v4r4/ic2924/info/INFOCENT.HTM**
  - For the Graphical Toolbox and PCML via the Java and AS/400 Toolbox for Java links
    - **http://publib.boulder.ibm.com/pubs/html/as400/v4r4/ic2924/info/java/rzahh/toolbox.htm**
    - **Then take links for Graphical Toolbox or PCML**

- AS/400 Toolbox for Java External website
  - **http://www.as400.ibm.com/toolbox/welcome.htm**

This publication may refer to products that are not currently available in your country.

Client Access, Client Access/400, AS/400, OS/400, and IBM are trademarks of the IBM Corporation in the United States or other countries or both.

Java is a trademark of Sun Microsystems, Inc.

NS/Router, and NS/Elite are trademarks or registered trademarks of Netmanage.

Microsoft, Windows, and the Windows 95/NT logo are trademarks or registered trademarks of Microsoft Corporation.

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