eProcess Developer's Guide

Release 5.0

Panagon eProcess Developer's Guide

- Using ADO to Query the eProcess Database

Quick Links

eProcess API Classes JavaDoc

Java UI Toolkit API JavaDoc

eProcess Developer's Guide Overview

This guide provides information for developers who are extending or customizing the Panagon eProcess out-of-the-box (OOTB) applications or who are creating custom workflow processing applications (sometimes called "work performers") for Panagon eProcess Services. Extending out-of-the-box applications may include creating Java™-, ASP-, or COM-based applications for Panagon eProcess Services, modifying the interface to eProcess services, creating new customized Java or HTML (ASP) Step or Launch Step Processors, or modifying and customizing the Java or HTML Step Processor and Launch Step Processor samples provided with the Panagon eProcess Toolkit.

Previous experience with Panagon Web Services, Panagon Web WorkFlo, or Panagon WorkFlo Services is not required. However, you must be familiar with the specific development concepts, technologies, and languages on which eProcess Services is based. If you are planning to develop Java applets or applications, you should also have experience using the Sun Microsystems Java Software Development Kit (SDK).

Topics covered in this overview include:

Overview of This Guide

Development information is provided in the following sections of this guide:

How to Use This Guide	Provides information on the documentation conventions used in the guide, printing the guide or individual topics, locating additional FileNET resources, and how to contact us to send comments or suggestions about the guide. A copy of the Panagon End User License Agreement is also included in this section.
What's New in 5.0	Provides information on What's New in 5.0, which describes the changes to the eProcess API (and other developer-related changes) from the previous Panagon eProcess 4.2.2 release to the current 5.0 release.

Understanding eProcess

Provides an overview of the Panagon eProcess Services architecture, the eProcess Services directory structure and the components you use while developing customized eProcess applications. The section also includes information on the referenced Web Services and Web WorkFlo files used for ASP development. Also described are the tools and languages needed for different development tasks.

Installing Panagon eProcess Toolkit

Describes the Panagon eProcess Toolkit, how to locate and install the eProcess Toolkit, the Toolkit contents, and how to access the eProcess Toolkit Web Services Help.

Developing with eProcess APIs and Samples

Provides information on:

Developing with eProcess APIs — describes the eProcess API Classes and API class relationships and calling sequences, developing for Java, developing for COM, and developing for ASP.

API samples — describes the API samples that are provided with the Panagon eProcess installation and that may be used as a starting point for developing your custom application (includes useful helper classes and information on how to compile and run the samples).

Customizing Java and HTML Applications

Provides information on:

Customizing Java and HTML Processors Overview — provides an overview of customizing Java and HTML (ASP) Step and Launch Step Processors.

Customizing Java Step and Launch Step Processors — includes an overview of the Java Step and Launch Step Processor Toolkit), describes issues on whether to deploy a custom Java Step Processor as an application or an applet, and describes how to use the Java Step Processor samples provided with the Toolkit, and working with the Java UI Toolkit interfaces and beans (to customize the user interface).

Customizing HTML (ASP) Step and Launch Step Processors — includes a description of the HTML Processor Toolkit files, modifying the sample HTML Step Processor, HTML Step Processor File relationships, and similarly with the HTML Launch Step Processor and the HTML Launch Step

Processor file relationships.

Deploying Step and Launch Step Processors — Provides procedures on deploying custom Java Step or Launch Processors, signing a JAR file (required for Java Processors), and adding custom Java and HTML Step or Launch Processors to the workflow.

Modifying Email Notification Messages

Provides information on modifying the email notification message files supported by Panagon eProcess Services, including how to enable SSL for email notification. The section contains reference information for each of the email notification message file types, including information on the localized language versions. How to set up a test mail server to test notifications is also described.

Additional Tips and Procedures

Provides a procedure on how to set up and use ADO to query the eProcess database, which can improve performance on queries on queues, rosters, or logs, where there are large amounts of data to retrieve.

Related eProcess API Documentation

This guide is only one part of the development documentation set you will need to develop applications or applets for Panagon eProcess Services. Additional related eProcess API documentation includes the *eProcess API Classes JavaDoc* and the *Java UI Toolkit JavaDoc* API reference documents, which are HTML-formatted documents containing API class reference information taken directly from the Java code (using the Sun Microsystems JavaDoc utility).

The *eProcess API Classes JavaDoc* and *Java UI Toolkit JavaDoc* API reference documents are included in the Panagon eProcess Toolkit.

Browser Compatibility

Some navigation components used in guide, like the expandable Table of Contents pane and the Search engine, work best in Microsoft Internet Explorer 5.0 (and later) and Netscape 6.0 (and later). If you experience problems viewing the table of contents or search results, verify that Javascript support is enabled in the browser you are using. If problems persist and you are unable to access the expandable contents pane, switch to an HTML version of the contents.

eProcess Required Tools and Languages

The tools and languages you use depend on your development goals. For example, if you are customizing the HTML Step Processors, your development requirements are much different than if you are developing or extending Java Step Processors. Regardless of the development goal, all eProcess Services development relies on a foundational understanding of the eProcess APIs.

Required tools and languages include:

- Java Development
- ASP and COM Development
- ASP.NET Open Client Development

Java Development

If you are developing or customizing Java Step Processors and/or Launch Step Processors for Panagon Web WorkFlo, or you are creating Java-based customized work object processing applications (also called work performers) for Panagon WorkFlo Services, you will need be familiar with the following languages, tools, and technologies:

- Java programming methodology
- Sun Microsystems Java Software Development Kit (SDK)
- Creating, signing, and deploying .JAR files
- ASP architecture and objects (for applets)
- Remote Method Invocation (RMI)
- VBScript
- HTML 4.0 (or higher)
- Panagon Web Services server-side includes (.ASP and .INC files)
- Panagon Web Services client-side Javascript libraries (.JS files)
- Panagon WorkFlo Services on HP-UX platforms, the HP JDK.
- For WorkFlo Services on AIX, the IBM AIX JDK.

Additionally, experience with an Integrated Development Environment (IDE), which can use the Rapid Application Development (RAD) components shipped as part of the Panagon eProcess Toolkit, might be useful. Examples of IDEs with RAD capability are Sun Microsystems Forte for Java, Visual Café, or Borland JBuilder.

An IDE helps speed up development time for projects that require visual user interfaces by allowing drag-and-drop functionality on the VWPanel container; however, an IDE is not required to develop applications using the Swing components.

ASP and COM Development

If you are developing or customizing the ASP components in Web WorkFlo, you will need to be familiar with the following languages, tools, and technologies:

- ASP architecture and objects
- Visual Studio development environment (Visual InterDev, Visual Basic, etc.)
- Javascript 1.1 (or higher) and VBScript
- HTML 4.0 (or higher)
- Panagon Web Services server-side includes (.ASP and .INC files)
- Panagon Web Services client-side Javascript libraries (.JS files)
- XML
- JiGlue COM Bridge

ASP.NET Open Client Development

If you are developing Panagon Open Client in an ASP.NET environment, you will need to be familiar with:

- ASP and ASPX architecture and objects
- JNI
- Visual Studio
- HTML 4.0 (or higher)
- XML
- Visual Basic (experience with VB programming with the ASP.NET SDK and Visual Studio, while helpful, is not an absolute requirement)
- Remote Method Invocation (RMI)
- ASP.NET
- Sun Microsystems Java Software Development Kit (SDK), if developing Java applets or applications.

For information on developing eProcess Open Client applications, see the "Developing for eProcess Open Client" section in the *Panagon Open Client Developer's Guide*.

eProcess API Class Relationships

The calling sequence diagrams demonstrate how the eProcess classes relate to each other and indicate which methods to call when using the eProcess Application Programming Interfaces (APIs).

These calling sequence diagrams are meant to be used with the detailed reference information found in the *eProcess API Javadocs* documentation. The *eProcess API Javadocs* documentation provide HTML-formatted class reference information taken directly from the Java source code of the classes (using the Sun Microsystems *Javadoc* utility).

Using the calling sequence diagrams

The calling sequence diagrams are separated, by function, into three functional areas:

- Runtime APIs
- Administration and configuration APIs
- Workflow definition APIs.

The diagrams are categorized to more easily present the calling sequences by functional activity. For example, when developing a system configuration utility you might want to refer primarily to the Administration and Configuration calling sequence diagram.

Additionally, this section includes some high-level information on the types of programmatic tasks supported by the APIs. See Working with eProcess APIs for more information. If you are using the eProcess APIs for Java development, refer to Java development, in the *Developing for Java* section of this guide. If you are using the APIs in COM development, refer to COM development, in the *Developing for COM* section of this guide.

Use the calling sequence diagrams, regardless of the development environment, to ensure you are using the preferred methods for calling these Application Programming Interfaces (APIs). The diagrams are not encyclopedic in detail; they show only the primary class relationships, and the preferred methods used to call them. Use the calling diagrams with the detailed class information found in the *eProcess API* reference documentation.

Some classes, like the interfaces and the VWAttachment and VWXMLUtil among others, do not appear on these calling diagrams. These classes can be created without first calling one of the primary classes. In most cases, however, the classes require that other API objects already have valid data or, as is the case with the VWAttachment class, require data from the IDM Objects to function correctly.

Using deprecated methods

The calling sequence diagrams do not show deprecated methods, since in all cases the deprecated methods have been replaced by a new, preferred method (which are shown). All deprecated methods have been marked in the reference documentation, and in almost all cases you will find a suggestion for

using an alternative method.

If this is the first time you are developing applications for Panagon eProcess, do not use the deprecated methods; these methods can be removed without notice. (You can find a listing of all of the current deprecated methods by selecting the **Deprecated** tab at the top of the each reference page from within the *eProcess API* JavaDoc documentation.)

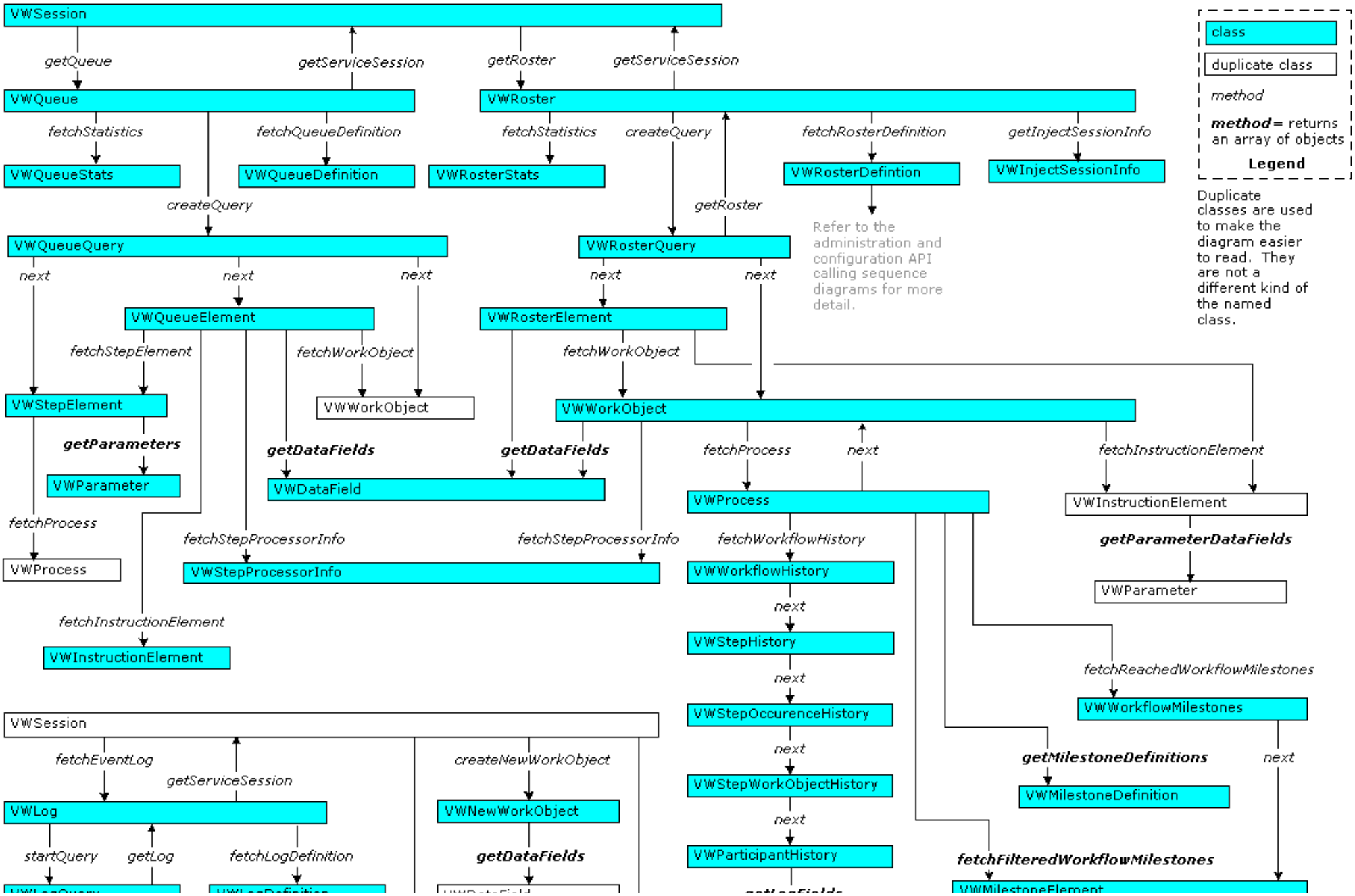
If you have already developed applications for Visual WorkFlo and you are upgrading an existing Visual WorkFlo application, your code should continue to function with the deprecated method(s). FileNET recommends updating the application to replace the deprecated methods with the new methods before upgrading to the next release. In addition, you should review the new classes and changes to existing classes and methods in the What's New in 5.0 topic.

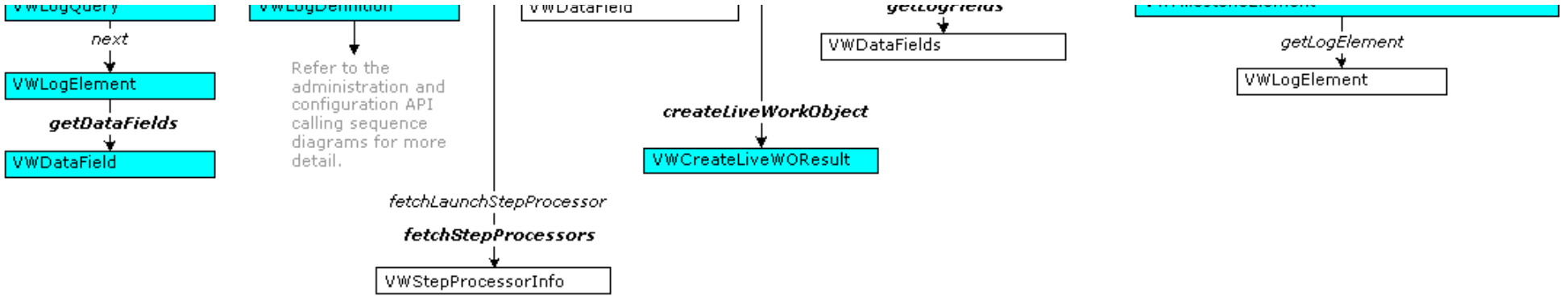
Tip The eProcess API documentation is located on the **Panagon Web WorkFlo and Toolkit CD**. Refer to the Install Panagon eProcess Toolkit topic for more information on locating the eProcess API documentation.

Runtime API Relationships

The diagram shows the calling sequences for the APIs that are useful retrieve work objects, query queues, retrieve Step Processor information, and other workflow related operations. See the Administration and Configuration API Relationships and Workflow Definition API Relationships topics for more information.

Note Duplicate classes appear in the diagram for readability and have no programmatic significance. The duplications simplify the connecting lines.

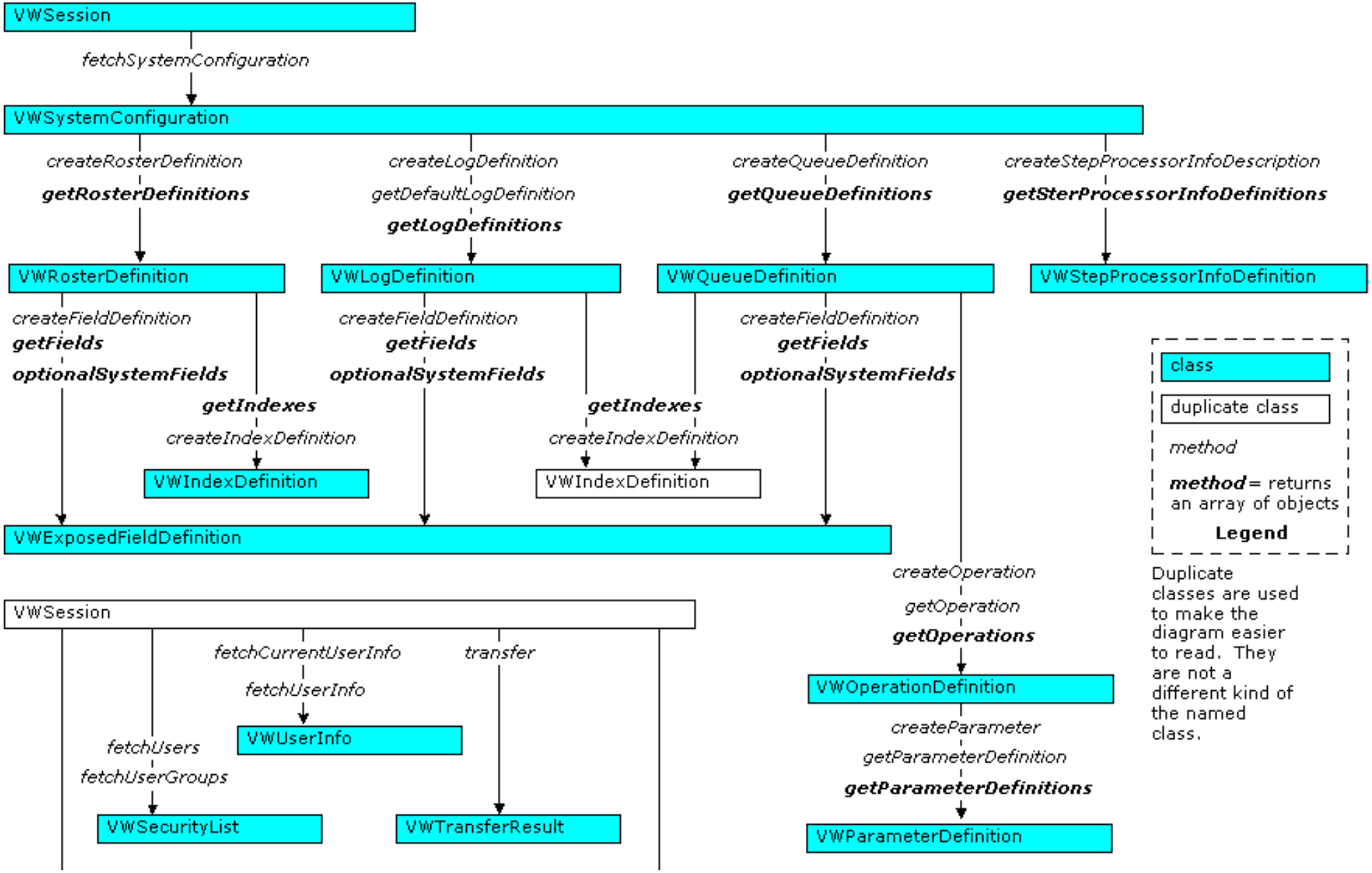



next
VWLogElement
getDataFields
VWDataField
Refer to the administration and configuration API calling sequence diagrams for more detail.
VWDataField
getLogFields
VWDataFields
createLiveWorkObject
VWCreateLiveWOResult
getLogElement
VWLogElement
fetchLaunchStepProcessor
fetchStepProcessors
VWStepProcessorInfo

Administration and Configuration API Relationships

The diagram shows the calling sequences for the APIs that are useful for configuring and administering Panagon eProcess. See the Runtime API Relationships and Workflow Definition API Relationships topics for more information.

Note Duplicate classes appear in the diagram for readability and have no programmatic significance. The duplications simplify the connecting lines.

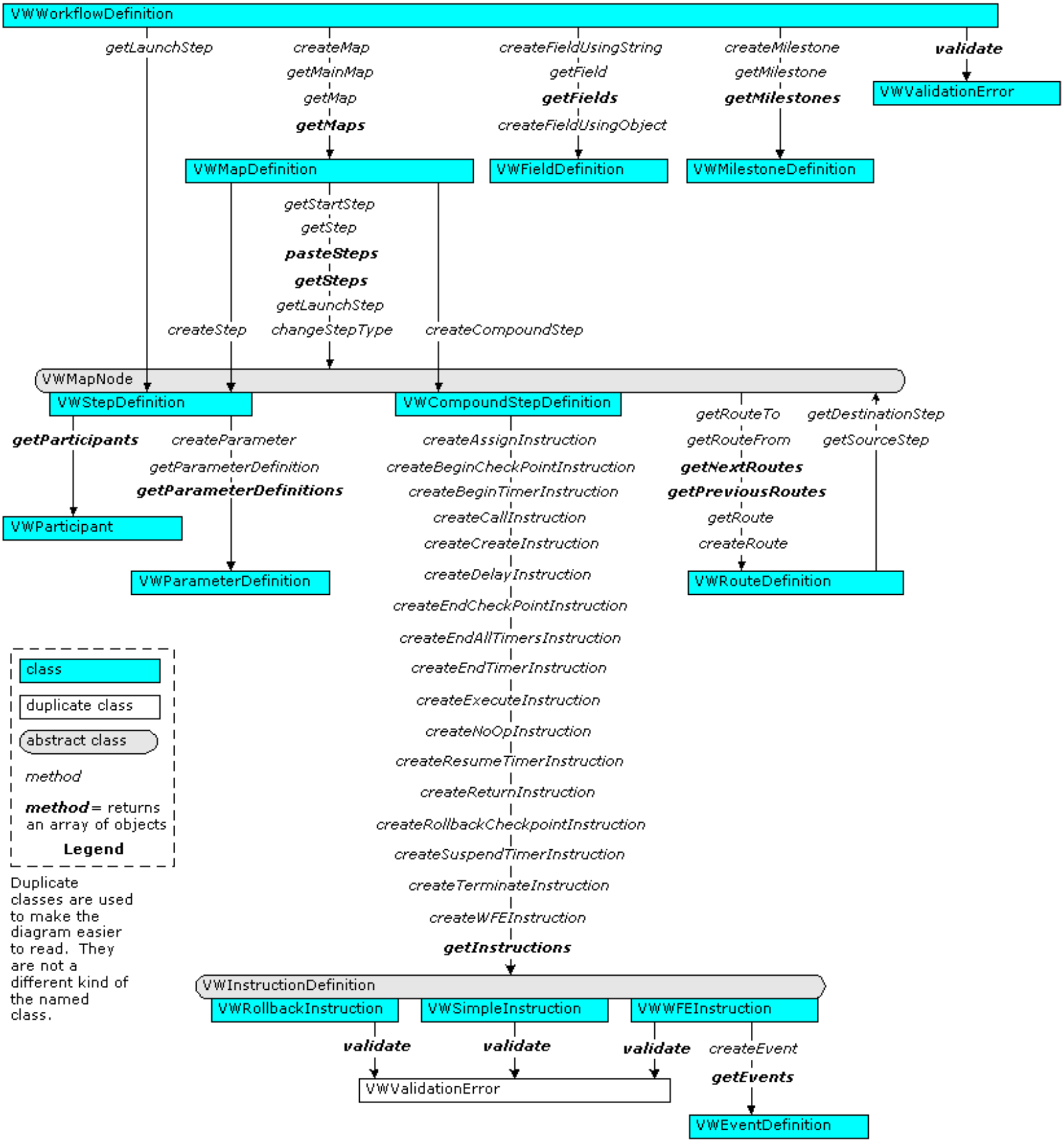



fetchSystemAdministration
VWSystemAdministration
createUserInfo
fetchUserInfo
VWUserInfo
fetchWorkflowSignature
VWWorkflowSignature
findField
getFieldDefinitions
VWFieldDefinition

Workflow Definition API Relationships

This diagram shows the calling sequences for the APIs that are useful for creating, modifying, and validating workflow definitions. See the Runtime API Relationships and Administration and Configuration API Relationships topics for more information.

Note Duplicate classes appear in the diagram for readability and have no programmatic significance. The duplications simplify the connecting lines.



Working with eProcess API Classes

Within the eProcess APIs there exist a limited number of classes which you must use to access a Panagon Workflo Services system. The primary classes are listed here and discussed below:

- VWSession
- VWWorkflowDefinition
- VWSystemConfiguration (once you have a valid session)
- VWSystemAdministration (once you have a valid session)

All other classes in the eProcess APIs require prior existence of one, or more, of these objects. In all cases, working within the eProcess API requires initial calls to one of these classes. The following sections provides a general introduction to the classes you can access using the APIs listed above. Refer to the *eProcess API* reference documentation for more detailed information.

VWSession

Constructed using the following method:

VWSession(username, password, router_URL)

Once you have a VWSession object, you can access the following classes to perform these general tasks:

Class	Functional description
VWQueue	With this object you can query queues and retrieves queue descriptions, information on the supported data fields, and step elements.
VWRoster	With this object you query rosters and retrieve information about the workflow participants.
VWLog	With this object you can log system and user-driven events to a log and retrieve log elements from the system.

Back to top.

VWWorkflowDefinition

Constructed using the following method:

VWWorkflowDefinition()

Once you have a VWWorkflowDefinition object to access the following classes to perform these general tasks:

Class	Functional description
VWMapDefinition	With this object you can access workflow steps and workflow map properties.
VWFieldDefinition	With this object you can access or set field definitions.
VWMilestoneDefinition	With this object you can access or set milestone definitions.

Back to top.

VWSystemConfiguration

Created using the following methods:

VWSession(username, password, router_URL)
VWSession.fetchSystemConfiguration()

Once you have a VWSession object, you can create a VWSystemConfiguration object to access the following classes to perform these general tasks:

Class	Functional description
VWRosterDefinition	With this object you can access and administer rosters.
VWLogDefinition	With this object you can access and administer event log definitions.
VWQueueDefinition	With this object you can access and administer queues.
VWStepProcessorInfoDefinition	With this object you can access and administer Step Processors.

Back to top.

VWSystemAdministration

Created using the following methods:

VWSession(username, password, router_URL)
VWSession.fetchSystemAdministration()

Once you have a VWSession object, you can create a VWSystemAdministration object to create users or access user information, using the VWUserInfo class.

Back to top.

Java Development

Java development for Panagon Web WorkFlo and WorkFlo Services does not require any particular development environment. All programming discussions and procedures for compiling the supplied samples contained in this guide assume that you are using the appropriate JDK (for details, see Configure the Java SDK).

When developing and testing Java applications for Panagon WorkFlo Services, you must have an installed and configured WorkFlo Services server with at least one initialized region to test the applications.

If you are developing customized workflow processing applications (sometimes called "work performers") for deployment on a client workstation, the **pw.jar** file, which contains all of the eProcess API classes, must reside on each client system. As part of the development process, the client workstation must also include any custom classes that a customized workflow processing application requires to run. Typically, you create a single Java ARchive (JAR) file that contains the eProcess API classes and your custom classes.

Note Developers who are used to earlier versions of the Visual WorkFlo APIs should keep in mind during development that there is not a one-to-one correspondence between each Visual WorkFlo API and eProcess API class. Generally, the two API sets have the same functionality; however, the eProcess APIs do not have the sequenced mode feature. For developers familiar with earlier eProcess API versions, see What's New in 5.0 for information on changes from the last Panagon eProcess release.

This section provides information on the following topics:

- Configuring the Java SDK on your development system to use the eProcess APIs class libraries.
- Starting an instance of the eProcess Router, which allows you to observe the RMI communication between your development system and a Panagon WorkFlo server.

Panagon WorkFlo Services requires that the Remote Method Invocation (RMI) mechanism be built into the Java environment. If you develop with Microsoft Visual J++™, install the Microsoft RMI patch before compiling your application code.

COM Development

The JiGlue (pronounced "JIG-LOO") COM Bridge acts as a translation layer between the eProcess Java APIs and COM. Using the JiGlue COM Bridge provides the ability to make calls similar to native Java statements for most Win32 programming tasks. Broadly speaking, the JiGlue COM Bridge provides the ability to instantiate COM objects using the Java APIs. Panagon Web WorkFlo uses the JiGlue COM Bridge whenever the Web WorkFlo ASP-based module must communicate with the WorkFlo Services server.

You can develop COM-based applications and utilities for both Panagon Web WorkFlo and Panagon WorkFlo Services. If you are developing against Panagon IDM Automation layer in a visual development environment (like Visual Basic), you can work with the FileNET-supplied JiGlue COM Bridge so your application can use the Java APIs; however, before using the JiGlue COM Bridge you must configure it on your development system; when deploying non-ASP-based applications you must configure the JiGlue COM bridge on each client system.

This section contains information on the following topics:

- Configuring the JiGlue COM bridge on your development system.
- Using the COM Bridge.
- Finding JiGlue member reference information.
- Understanding how JiGlue converts Java data types to COM equivalents.
- Adding JiGlue to Visual Basic.
- Using JiGlue in an ASP environment.

Installing Panagon eProcess Toolkit

Panagon eProcess Web WorkFlo and WorkFlo Services ships with an eProcess toolkit for developers. The *Panagon eProcess Toolkit* consists of the API classes, API samples and utilities, developer guide documentation, and API reference documentation for both the *eProcess Java APIs* and the Java *UI Toolkit APIs*.

This topic describes:

- Panagon eProcess Toolkit Installation
- Panagon eProcess Toolkit Contents
- Step Processor and Launch Step Processor Toolkit Files

Panagon eProcess Toolkit Installation

Install the Toolkit development files by performing the following steps:

1. Insert the *Panagon Web WorkFlo and Toolkit CD* into the CD-ROM drive on your system.
2. On the CD, locate the **\Developer Files** folder.
3. Copy the **\Developer Files** folder, and all the sub-directories, to your local drive. Maintain the relative directory structure of the directories.

Tip While no location requirements exist for the toolkit you should copy the directory structure to the root directory on your development system. Adding the directory to the root speeds up compile time for Java applications, makes registering files easier, and makes finding the appropriate reference documentation much easier.

4. Change the file attributes for all of the files in the newly created directory structure from Read-only to Read/Write.
5. If you have not already done so, install the Java Software Development Kit (SDK). For details on which JDK to use for your development environment and how to install and configure it for eProcess, see the Configure the Java SDK topic in this guide.
6. Configure the Java SDK to use the appropriate files. If you are developing COM-based application, configure the JiGlue COM Bridge on the development system.

Refer to the eProcess Required Tools and Languages topic for more information on the languages, tools, and technologies you must be familiar with before developing applications for Panagon Web WorkFlo and WorkFlo Services.

Panagon eProcess Toolkit Contents

This section describes the Panagon eProcess Toolkit directory structure and contents. If you followed the installation instructions (above) the eProcess Toolkit files should be located on your development system in a directory structure contained in the **\Developer Files** directory. By default the **\Developer Files**

directory structure contains the following directories and files:

Directory	Contents
\Developer Files	Contains the following development files: ● **pw.jar**: contains the eProcess API runtime classes. ● **vwpanel.jar**: includes the information and classes necessary to use Rapid Application Development (RAD), drag-and-drop functionality in a Integrated Development Environment (IDE), such as Visual Cafe, JBuilder, Forte, etc. ● **jiglue.dll**: dynamic link library used to facilitate COM to Java communication (also called the JiGlue COM Bridge). ● **jiglue.tlb:** the JiGlue type library. **Note** The eProcess Swing-based runtime control components are included in the **pw.jar** file. However, if you want to use drag-and-drop RAD programming in your IDE to customize the user interface with these Swing-based eProcess controls, you must add **vwpanel.jar** to your IDE.
Developer Files\ Documentation	Includes the following directories: ● **\Developer Guide**: contains the *Panagon eProcess Developer's Guide* (this document) HTML and image files. Open **\Developer Guide\eprocess_dg_start.htm** to begin viewing the document. ● **\JavaDoc\JavaAPI**: contains the *eProcess API* JavaDoc documentation for the eProcess API classes. Open **\JavaDoc\JavaAPI\index.html** to begin viewing the reference documentation. ● **\JavaDoc\UI_Toolkit_API**: contains the *UI Toolkit* JavaDoc documentation for the User Interface Toolkit Swing-based eProcess controls. Open **\JavaDoc\UI_Toolkit_API\index.html** to begin viewing the reference documentation. **Tip** The *UI Toolkit* JavaDoc documentation is useful if you plan to create or extend Java Step Processors or Launch Step Processors using the Swing-based eProcess controls that come with the Panagon eProcess Toolkit.

Developer Files\ Samples	Includes the following directories: • **\API**: contains Java sample files that demonstrate how to create, launch, complete, display, and save log information for a workflow. • **\vwpanel**: contains the Java Step Processor samples files (in the **\samplestep** subdirectory) and Java Launch Step Processor sample files (in the **\samplelaunch** sub-directory) along with the image files needed to run the compile and run the samples. • **\Utils**: contains a Java class that allows you to view or remove entries from the RMI registry.

Step Processor and Launch Step Processor Toolkit Files

To modify the Step Processors or Launch Step Processors a developer must first know how to use the eProcess APIs. One way of looking at the Step Processors and Launch Step Processors is to view them as specialized applications that make extensive use of the eProcess APIs, existing Panagon Web Services components, and the JiGlue COM Bridge (HTML versions only).

This section separately lists the toolkit files for both ASP (HTML) and Java Processors:

- Toolkit files for ASP (HTML) Processors
- Toolkit files for Java Processors

Toolkit files for ASP processors

Some eProcess Toolkit files for ASP development were installed on your Web WorkFlo server under the following directory structures (if the default location was selected during Panagon Web Services and Web WorkFlo installations):

Note The directories in the following table are relative to the **<drive>...\Program Files\FileNET\IDM\Web\IDMWS** directory structure on the Web WorkFlo server.

ASP files	Location
Step Processor files:	**\Redist\WF_Step\html**
Launch Step Processor files:	**\Redist\WF_Launcher\html**
Shared files:	**\Redist\WF_Html_Toolkit**

These files are used by the Step Processors and Launch Step Processors shipped with Panagon Web WorkFlo. You need not install additional files to see these ASP eProcess Toolkit files operate.

Refer to the HTML Processor Toolkit Overview topic for more information on understanding the default structure of the ASP-based Step Processor and Launch Step Processor files. The HTML processors use the JiGlue COM Bridge and Panagon Web Services Javascript objects. You must understand how to use these development elements to modify the HTML processors successfully.

Toolkit files for Java processors

The Java Step Processor (**<drive>...\Developer Files\Samples\vwpanel\samplestep**) and Launch Step Processor (**<drive>...\Developer Files\Samples\vwpanel\samplelaunch**) samples, along with the UI Toolkit JavaDoc documentation (**<drive>...\Developer Files\Documentation\\JavaDoc\UI_Toolkit_API**), constitute the Java Processor development toolkit.

Additionally, ASP containers for the Java applet versions of the Step Processor and Launch Step Processors are provided, as follows:

Note The directories in the following table are relative to the **<drive>...\Program Files\FileNET\IDM\Web\IDMWS** directory structure on the Web WorkFlo server.

Applet container files	Location
Step Processor container file:	**\Redist\WF_Step\tabbed\step_main.asp**
Launch Step Processor container file:	**\Redist\WF_Launcher\tabbed\launcher_main.asp**

Locate Panagon Web Services Toolkit Help

Panagon Web WorkFlo uses the Javascript API objects, ASP files, and in some cases, the IDM Objects provided in Panagon Web Services and Panagon IDM Desktop.

To access, or interact with, content stored in Panagon Content Services or Panagon Image Services libraries, you must be familiar with the Javascript API objects and IDM Objects. (See the Panagon eProcess Services Architecture topics for more information on how these objects interact with the eProcess APIs.)

The *Panagon Toolkit Help* system provides detailed reference and programming information on the Panagon Toolkit COM IDM Objects and controls, including samples and *Working with. . .* topics. Familiarize yourself with the development information available in the *Panagon Toolkit Help* system.

Access *Panagon Toolkit Help*, which contains the developer's documentation for Panagon Web Services, by performing the following steps:

1. On your eProcess Services server, click **Start**.
2. Select **Programs** > **FileNET Panagon IDM** > **Help For Developers**. The *Panagon Toolkit Help* system displays.

Access the Panagon Web Services programming help by performing the following steps, while still in the *Panagon Toolkit Help* system:

1. From the *Panagon Toolkit Help* **Contents** tab, locate the *Web Services* book (near the bottom of the topic list).
2. Highlight and double-click the *Web Services* book.

The following table lists the kind of information you can expect to find in the Web Services help:

Title	Content description
Panagon Web Services Overview	Includes detailed information about the Panagon Web Services architecture and objects about a Panagon Web Services server before Panagon Web WorkFlo is installed.
Developing for the Web	Includes information necessary for developing a web application for Panagon Web Services. Topics describe Panagon Web Services toolkit components, where to find them, and how to use them. Some topics discuss how to create ASP files that perform specific functions. *Working with. . .* topics discuss how to work with four Javascript API objects: IDMWSC_Document, IDMWSC_Folder, IDMWSC_Library, and IDMWSC_StoredSearch. Most topics include basic sample code to demonstrate how to use the objects.

Reference	Includes reference information for all Web Services API Javascript objects, methods, properties, events, and addressable Uniform Resource Locators (URLs).
Using Web Services Sample Pages	Includes information on using and modifying the IDM Web Services sample .ASP and .HTM pages.

What's New in 5.0

This section summarizes the API and API-related changes made between Panagon eProcess Release 4.2.2 and Release 5.0.

Topics covered in this section include:

- New API Classes
- Modified API Classes
- Modified Java UI Toolkit Beans and Interfaces
- Deprecated Class Members: Methods and Static Values
- Miscellaneous Development/API-Related Changes. These are changes made not to any APIs, but which may affect API usage.
- Documentation Changes

Notes

1. The major feature change for the eProcess 5.0 Release, the addition of the Analysis and Reporting Engine, required little change to the eProcess API, except for the addition of flags for the VWSystemAdministration class to support future uses.
2. eProcess integration with Open Client (in addition to the current ActiveX client) for use with the IDM .NET 3.2 Open Client will initially be provided for use with the eProcess 4.2.2 release. For information on using eProcess in an environment that includes both Open Client and ActiveX clients, see Miscellaneous Development/API-Related Changes below. For information on developing eProcess applications for IDM .NET-based Open Client, see the "Developing for eProcess Open Client" section of the *Open Client Developer's Guide*.

New API Classes

One new public class, VWAttributeInfo, has been added to the eProcess API to support the use of attributes and also to support ActiveX/Open Client coexistence (future use).

Class	Method Included	Class or Method Description
VWAttributeInfo	-------	This class enables users to associate properties they define with VWOperationDefinition, VWQueueDefinition, VWRosterDefinition, VWStepDefinition, VWSystemAdministration, VWSystemConfiguration, and VWWorkflowDefinition objects. An additional future use of this class will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available (see Miscellaneous Development/API-Related Changes below).
	void deleteField (String fieldName)	Deletes the field entry that is specified by input field name.
	String[] getAttributeNames ( )	Gets the array of attribute names for this VWAttributeInfo object.

	int getFieldType (String fieldName)	Returns the field type of the field value that corresponds to the input field name.
	Object getFieldValue (String fieldName)	Gets the object value for the input field name.
	boolean isFieldArray (String fieldName)	Indicates whether the field is an array.
	void setFieldValue (String fieldName, Object fieldValue)	Creates, deletes, or updates a VWAttributeInfo field value. The field names are labels for fieldValues.

Back to top

Modified API Classes

The following table summarizes the new methods added for this release.

Class	Members Added	Description	Reason for Change
VWActionType	ACTION_TYPE_DB_EXECUTE = 37	Action type for a DB_exec instruction.	Future support for a Database System Instruction feature.
VWActionType	ACTION_TYPE_LOG = 38	Action type for a logging instruction.	Support for design-time customization of logging events.
VWCompoundStepDefinition	VWInstructionDefinition createLogInstruction (String theEventType, String theEventText)	Creates a new instruction that will be added to the end of the list of instructions currently in this compound step. Given the input event, the input message will be logged.	Support for design-time customization of logging events.
VWOperationDefinition, VWQueueDefinition, VWRosterDefinition, VWStepDefinition, VWSystemAdministration, VWSystemConfiguration, and VWWorkflowDefinition	VWAttributeInfo getAttributeinfo ( )	Returns theVWAttributeInfo object for this class.	Attribute support for these objects. See New Classes.

VWOperationDefinition, VWQueueDefinition, VWRosterDefinition, VWStepDefinition, VWSystemAdministration, VWSystemConfiguration, and VWWorkflowDefinition	void setAttributeInfo (VWAttributeInfo)	Sets the VWAttributeInfo object for this class.	Attribute support for these objects. See New Classes.
VWQueue	QUERY_GET_NO_ SYSTEM_FIELDS = 1024 QUERY_GET_NO_TRANSLATED _SYSTEM_FIELDS = 2048	New static values to provide more effective query filtering.	Enhance queries to filter flexibly on system fields.
VWRoster	QUERY_GET_NO_ SYSTEM_FIELDS = 1024 QUERY_GET_NO_TRANSLATED _SYSTEM_FIELDS = 2048	New static values to provide more effective query filtering.	Enhance queries to filter flexibly on system fields.
VWSession	ATTRIBUTES_SYSTEM = 1 ATTRIBUTES_REGION = 2	New static values to flag whether to fetch system-wide or isolated region-specific attribute information for the current logged-on session.	Used to specify either system-wide or isolated region-specific attribute information.
VWSession	DATABASE_ORACLE= 1 DATABASE_SQL2000 = 2	New static values to define the database environment as database configuration types.	Future support for a Database System Instruction feature.
VWSession	WEBAPP_NONE = 0 WEBAPP_WORKPLACE = 1 WEBAPP_WEB_WORKFLO = 2 WEBAPP_OPEN_CLIENT = 3 WEBAPP_CUSTOM = 100	New static values (as shown) to define the web application IDs. **Note:** Values 4-99 are reserved for FileNET use. Values 101-nnn are available for customer use.	Used to specify the web application. An additional future use will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.

VWSession	VWAttributeInfo fetchAttributes(int nFlag)	Fetches from the workflow server system-wide or region-specific attribute information for the current logged-on session.	Useful for a customization that creates system-wide or region-specific attribute information via VWSystemAdministration or VWSystemConfiguration objects, respectively.
VWSession	int getDatabaseType()	Returns the database configuration type of the current database.	Future support for a Database System Instruction feature.
VWSession	int getDefaultWebApplication ( )	Returns the application environment type.	Gets the default web application ID for a logged-on session. The web application id enables the workflow system to use web application-specific program implementations.
VWSession	String getServerName ( )	Returns the name of the server for this session.	Enables an API user to get the server name at runtime.
VWSession	void setDefaultWebApplication (int theWebApplicationId)	Stores the application environment type as one among: NONE, WORKPLACE, WORKFLO, OPEN_CLIENT, or CUSTOM.	Sets the default web application ID for a logged-on session. The web application id enables the workflow system to use web application-specific program implementations.
VWStepProcessorInfo	Hashtable getLocations ( )	Gets an array of step processor URL locations, which are typically URLs of ASP or JSP pages.	Future use of this method will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.
VWStepProcessorInfoDefinition	VWStepProcessorInfoDefinition (int, String, int, Hashtable, int, int, String)	Constructor to specify the URL locations of stepProcessors.	Future use of this method will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.

VWStepProcessorInfoDefinition	String getLocation (int)	Gets the URL location associated with this step processor info object, for one Web Application. The location may be a URL of the asp page.	Future use of this method will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.
VWStepProcessorInfoDefinition	Hashtable getLocations ( )	Gets the URL location associated with this step processor info object. The location may be a URL of the asp page.	Future use of this method will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.
VWStepProcessorInfoDefinition	void setLocations (Hashtable theLocations)	Sets the URL location associated with this step processor info object.	Future use of this method will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.
VWSystemAdministration	Commit ( )	Saves the session timeout interval, system flags, statistics consolidation interval, and attribute information to the workflow server database.	Future use of this method will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.
VWSystemAdministration	getSessionTimeOut( )	Gets the number of minutes before a client / server session timeout.	Support for customization of the Session timeout period, rather than relying on the system default of 20 minutes.
VWSystemAdministration	void setSessionTimeOut(int)	Sets the number of minutes before a client / server session timeout.	Support for customization of the Session timeout period, rather than enforcing a system default of 20 minutes.

VWSystemAdministration	SYSTEM_WIDE_FLAG_ CONSISTENT_RETURN = 0x200	Indicates that returns will be consistent with the documented behavior of the return system instruction.	Future support for Analysis and Reporting features.
VWSystemAdministration	SYSTEM_WIDE_FLAG_ NO_ANALYSIS_ENG = 0x100	Indicates that the Analysis Engine should be disabled on the server.	Future support for Analysis and Reporting features.
VWWorkflowDefinition	LVALUE = 0	Provides a standard static value for locating the pre-assignment values of an assignment array, in common with step definitions and event definitions.	Used to support pre-assignment and post-assignment two-dimensional arrays.
VWWorkflowDefinition	RVALUE = 1	Provides a standard static value for locating the post-assignment values of an assignment array, in common with step definitions and event definitions.	Used to support pre-assignment and post-assignment two-dimensional arrays.

Back to top

Modified Java UI Toolkit Beans and Interfaces

The *IVWParameterConstants* interface's fields for session and web application-related operations were modified for this release as shown in the following table.

Interface/Bean	Fields Deprecated	Fields Added, Deleted, or Replaced	Reason for Change

IVWParameterConstants		The fields shown below were added, deleted, or replaced existing fields. The field string constant values shown specify the web application IDs.	Provide web application IDs. An additional future use will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available (see Miscellaneous Development/API-Related Changes below).
IVWParameterConstants	PRODUCT	WEBAPP Value: webApp	Same as above.
IVWParameterConstants	BRIGHTSPIRE_PRODUCT	WEBAPP_WORKPLACE Value: WORKPLACE	Same as above.
IVWParameterConstants	PANAGON_PRODUCT	WEBAPP_WEB_WORKFLO Value: WEB_WORKFLO	Same as above.
IVWParameterConstants	OPEN_CLIENT_PRODUCT	WEBAPP_OPEN_CLIENT Value: OPEN_CLIENT	Same as above.
IVWParameterConstants		VIEWER Value: VIEWER	New field added. Enables the applet to determine which document viewer to display (ActiveX or Java).
IVWParameterConstants	CUSTOM_PRODUCT	Deleted (not replaced).	FileNET eProcess does not currently support a sample custom product.

Back to top

Deprecated Class Members: Methods and Static Values

The following table summarizes API class members that have been deprecated as of eProcess Release 5.0.

Note For information about any specific deprecation, refer to the "Deprecated Methods" section of the *eProcess API JavaDoc* documentation; for information on *eProcess API JavaDoc* documentation, see eProcess API Class Relationships.

Class	Member deprecated	Replacement	Reason for Change

VWQueue	QUERY_GET_SYSTEM_FIELDS = 256 QUERY_GET_TRANSLATED_SYSTEM_FIELDS = 512	Not Replaced. This value is the default.	To provide for inclusion or exclusion of system fields or translated system fields for query filtering.
VWRoster	QUERY_GET_SYSTEM_FIELDS = 256 QUERY_GET_TRANSLATED_SYSTEM_FIELDS = 512	Not Replaced. This value is the default.	To provide for inclusion or exclusion of system fields or translated system fields for query filtering.
VWSession	PRODUCT_BPS	Replaced by WEBAPP_WORKPLACE; use with setDefaultWebApplication(int).	Future use of the replacement member will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.
VWSession	PRODUCT_CUSTOM	Replaced by WEBAPP_CUSTOM; use with setDefaultWebApplication(int).	Future use of the replacement member will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.

VWSession	PRODUCT_OPEN_CLIENT	Replaced by WEBAPP_OPEN_CLIENT; use with setDefaultWebApplication(int).	Future use of the replacement member will be to support ActiveX/Open Client coexistence when eProcess 5.0 integration with Open Client becomes available.
VWSession	PRODUCT_PW	Replaced by WEBAPP_WEB_WORKFLO; use with setDefaultWebApplication(int).	Helps manage more flexible environment specifications in multiple web application system configurations.
VWSession	setAuditState()	Replaced by inserting an edited copy of an fnlogging.properties file in the <JRE>\lib directory. See this method in the eProcess API Javadoc for details.	Support for the Sun 1.4 JRE.
VWSession	setProductId(int)	Replaced by setDefaultWebApplication(int).	More flexible support for multiple Web Application system configurations.
VWStepProcessorInfo	getLocation()	Replaced by getLocation(int).	To allow specification of one among multiple web application locations for a single step processor.
VWStepProcessorInfoDefinition	VWStepProcessorInfoDefinition (int theId, int theProcessorType, String theName, int theAppType, String theLocation, int theWidth, int theHeight, String theApp)	Replaced by VWStepProcessorInfoDefinition (int theId, int theProcessorType, String theName, int theAppType, Hashtable theLocation, int theWidth, int theHeight, String theApp).	To allow specification of multiple web application locations for a single step processor.

VWStepProcessorInfoDefinition	getLocation()	Replaced by getLocation(int).	To allow specification of one among multiple web application locations for a single step processor.
VWStepProcessorInfoDefinition	setLocation(String)	Replaced by setLocations (Hashtable).	To allow specification of one among multiple web application locations for a single step processor.
VWSystemAdministration	setSystemWideFlags(int theFlags)	Replaced by setSystemFlags (int theFlags). This replacement does not save system flags to the workflow server.	More flexible flag setting replacement.

Back to top

Miscellaneous Development/API-Related Changes

The following are changes made for the eProcess 5.0 Release that were not directly made to any APIs, but which can affect how the APIs are used:

- Support for multi-dimensional arrays for the JiGlue COM Bridge. This change enables API methods that take multi-dimensional arrays as parameters to be called for JiGlue (previously JiGlue only supported one-dimensional arrays). Following are API methods (in addition to the new Attribute-related methods described above) that take multi-dimensional arrays as parameters and can now be called for JiGlue:

```
VWEventDefinition
     String[][] getAssignments()
     void setAssignments(String[][] theAssignments)


VWCompoundStepDefinition
     VWInstructionDefinition createCreateInstruction(String theWorkClassName,
String[][] theFieldAssignList)
     VWInstructionDEfinition createAssignInstruction(String[][] assignPairs)
```

- The Java Runtime Environment (JRE) has been upgraded for the eProcess 5.0 release for certain platforms. That is, for the eProcess 5.0 release:
  - For the Web client, use the Sun Microsystems JDK 1.4.0.
  - For the Web server, use the Sun Microsystems JDK 1.3.1.
  - For the WorkFlo Services server on a Windows 2000 platform, use both the Sun Microsystems JDK 1.3.0 and 1.3.1:
    - WorkFlo server: requires JRE 1.3.0.
    - NT Services: requires JRE 1.3.1.

- For WorkFlo Services on Solaris, use the Sun Microsystems JDK 1.4.0.
- For WorkFlo Services on HP-UX platforms, use the HP JDK 1.3.1.
- For WorkFlo Services on AIX, use the IBM AIX JDK 1.3.0 (for AIX 4.3 and higher).
- For Panagon Open Client, the Sun JDK 1.3.1 is used on the web server, JDK 1.4.0 on the client (for information on developing eProcess Open Client applications, see the "Developing for eProcess Open Client" section in the *Panagon Open Client Developer's Guide*).

  Note Versions used may be updated. For the latest information on which versions of software (including JDKs) are supported for Panagon eProcess Web Services, Web WorkFlo and WorkFlo Services, see the *eProcess Compatibility/Dependency/Server Matrix* document, located at http://www.css.filenet.com web site (go to **Product Info** > **Panagon** > **eProcess** > **Compatibility and Dependency**).

- As of the eProcess 5.0 release, before you can run a Step or Launch Step Processor application/applet (such as the sample Java Step Processor, `VWSampleStepApplication`, included with the Panagon eProcess Toolkit), you will need to use the Configuration Console to specify the web application defined for VWService — for details on configuring the server for a web application, see "Managing VWServices" in the *eProcess Configuration Console Help*.

  You will then specify the web application when you run your application. For instance, to run the sample Java Step Processor, `VWSampleStepApplication`, you would specify as a parameter: `webApp=WEB_WORKFLO` (previously, you would have specified: `product=PANAGON`). For example:

  ```
  javaw -classpath .;C:\Developer Files\samples\vwpanel\sample.jar;C:\<your directory>\pw.jar;%CLASSPATH% samples.vwpanel.samplestep.VWSampleStepApplication /hostName=<host name> /routerNames=vwrouter /routerPorts=1099 /idmPort=1099 /webApp=WEB_WORKFLO /queueName=InBox /wobNumber=B8D270F79636E1489BA9EC2DD9F7E950
  ```

  For details, see the Run the Sample Step Processor Application step in the Building and Deploying the Sample Java Processors procedure.

- Five new pseudo-fields have been added, so that a program module can more conveniently perform query filtering for work items, thereby limiting the need for maintaining user fields for common filtering tasks. The pseudo-fields are: F_StepName, F_StepDescription, F_InstrSheetName, F_LastErrorNumber, and F_LastError.
- As noted above, eProcess integration with Open Client (in addition to the current ActiveX client) for use with the IDM .NET 3.2 Open Client will initially be provided for use with the eProcess 4.2.2 release. Using Open Client with eProcess 4.2.2 in an environment that includes both Open Client and ActiveX clients requires that you use a separate web server, isolated region, router, and preferences for ActiveX and similarly, a separate web server, isolated region, router, etc. for Open Client. In other words, co-existence is not available with eProcess 4.2.2.

  The eProcess 5.0 release however, will support ActiveX and Open Client co-existence on the same web server, isolated region, and router at such time as Panagon Open Client is released for use with eProcess 5.0. For information on developing eProcess applications for ASP.NET-based Open Client, see the "Developing for eProcess Open Client" section of the *Open Client Developer's Guide*.

Back to top

Documentation Changes

The following documentation changes have been made since the last release:

- The *Panagon eProcess Developer's Guide* (this guide) has been updated for new features and restructured with a new Table of Contents. The index and search capabilities have been expanded. Code samples and examples have been updated and corrected.
- The *eProcess API JavaDoc* documentation has been updated for the changes indicated in this document. In addition, in many cases, class and method descriptions have been revised and corrected; additional and revised code samples and examples of usage have been provided.

Back to top

Java UI Toolkit Overview

The *UI Toolkit APIs* (interfaces, classes, and beans) are designed to help you develop Swing-based applications that use the eProcess APIs. To use these interfaces and classes, you must be familiar with the eProcess APIs. These interfaces, support classes, and beans are meant to make user interface development easier once you understand the programmatic requirements of the eProcess APIs.

This overview information, and the information found in the UI Toolkit API documentation, is useful if you are using these Swing components and controls to extend the interface of the Java Step Processors or Launch Step Processors. You can use these Swing components in one of two ways: (1) explicitly coding using these components in a text-based editor, or (2) drag-and-drop development in an IDE capable of supporting RAD. For drag-and-drop development you must add the vwpanel.jar file to the IDE.

While runtime versions of these classes are included in the **pw.jar** file, the necessary support files for drag-and-drop development are packaged in the vwpanel.jar file. Refer to Install Panagon eProcess Toolkit for more information on accessing the *UI Toolkit* APIs reference documentation.

Interfaces

Name	Description
IVWAppLauncher	Interface that must be implemented by all applications that need to share session information with child applications.
IVWFrameInterface	Interface that specifies which methods the dialogs support. This is a parent class for the IVWAppLauncher and IVWLaunchableApp interfaces.
IVWLaunchableApp	Interface that must be implemented in child applications that need to share session information with the parent application.
IVWPanelComponent	Implements required methods for all components for which VWPanel or VWLaunchPanel beans act as a container.
IVWParameterConstants	Interface that contains String versions of all of the necessary parameters for launching and running applets and applications.
IVWStepProcessor	Interface that should be implemented by any Step Processor application that allows the toolkit to modify its title.

IVWStepProcessorComponent	Interface that should be implemented by all components implemented as a Step Processor bean. Additionally, the VWPanel or VWLaunchPanel bean can contain these objects.

Support Classes

Name	Description
VWDriverFrame	Provides a container for customized frame implementations (extends the java.awt.Frame class).
VWPanelComponentInfo	Implements required interfaces for all components contained by the VWPanel. Retrieves information about the specified parameter from the step element and saves the changes to the step element.
VWSessionInfo	Provides a container for session related information (like host name, router name, router ports, and other necessary session data).

Beans (containers and controls)

See the UI Toolkit Parameters topic for information on the parameters supported by these beans.

Name	Description
VWAttachmentPanel	Provides the means for listing attachments associated with the current step. The toolbar provides the Content Services and Image Services functionality. This bean provides the only programmatic means to interact with the Content Services and Image Services within the UI Toolkit APIs. No other UI Toolkit APIs allow interaction with attachments.
VWButton	Provides support for some Step Processor operations, like save, complete, cancel, or reassign.
VWCheckBox	Provides a means for exposing the state of a Boolean data field. If the field is editable, the user changes the state by clicking on the control.

VWComboBox	Displays a list of responses. Calling the doSave() method saves the selected response.
VWFieldPanel	Displays the data fields exposed at the step. The panel allows users to modify the values based on the specified workflow definition mode.
VWLabel	Provides a means for displaying read-only parameter data.
VWLaunchPanel	Provides a container for other controls. This class is used to build a Launch Step Processor. The class creates the workflow, initializes the other components, and serves as an ActionListener for the button press events from the VWButton objects.
VWMilestonePanel	Provides a read-only milestone list.
VWPanel	Provides a container for other controls. This class is used to build a Step Processor. The class retrieves the VWStepElement, initializes the other components, and implements an ActionListener for the button press events from the VWButton objects.
VWTabbedPane	Provides a tabbed control for displaying the attachments, fields, and workflow group beans. In addition, the control allows an end user to select a response and enter his/her comments.
VWTextArea	Provides a multi-row control for displaying step instructions, or allowing the user to enter his/her comments.
VWTextField	Provides a single-row control for displaying field data.
VWWflGroupPanel	Provides a dialog box for displaying the Workflow Group parameters exposed at this step. The panel allows users to modify the contents of the groups.

All of the component beans have an associated information class. The information classes are named using the convention of adding BeanInfo to the bean name. For example, the information class for VWButton is named VWButtonBeanInfo. The information classes describe the associated beans, and the information classes support the following methods:

- getBeanDescriptor(): The method returns the BeanDescriptor for the class.
- getDefaultPropertyIndex(): The method returns the zero-based index value of the default property of the bean.

- getIcon(): The method returns the image used in the toolbar or a form.
- getPropertyDescriptors(): The method returns an array of PropertyDescriptor objects which represent the properties on the bean.

UI Toolkit Parameters

This topic lists the system fields supported by the different UI Toolkit beans (controls). While you can add other system fields to your customized Java Step Processor and Launch Step Processors, system field information other than that listed here is ignored.

The "PARAM_" constants are defined in the IVWPanelComponent class. The "PARAM_" constants map to the associated "F_" string value (shown below).

Some UI Toolkit controls support only one parameter, like the VWComboBox control, or a small set of the parameters, like the VWButton control. The IVWPanelComponent interface provides support for creating user-defined fields for the VWCheckBox, VWLabel, VWTextArea, VWTextField controls.

The following sections list the parameters supported by each control and are separated by control type.

The VWButton control supports the following parameters:

- F_Cancel (PARAM_CANCEL)
- F_Complete (PARAM_COMPLETE)
- F_Help (PARAM_HELP)
- F_Reassign (PARAM_REASSIGN)
- F_Return (PARAM_RETURN)
- F_Save (PARAM_SAVE)
- F_Status (PARAM_STATUS)

The VWComboBox control supports the following parameter:

- F_Responses (PARAM_RESPONSES)

The VWCheckBox, VWLabel, VWTextArea, VWTextField controls support the following parameters:

- User-defined fields
- F_Comment (PARAM_COMMENT)
- F_DateReceived (PARAM_DATE_RECEIVED)
- F_DeadLine (PARAM_DEADLINE)
- F_ExceptionDescription (PARAM_EXCEPTION_DESCRIPTION)
- F_ExceptionMap (PARAM_EXCEPTION_MAP)
- F_LaunchDate (PARAM_LAUNCH_DATE)
- F_OperationName (PARAM_OPERATION_NAME)
- F_Originator (PARAM_ORIGINATOR)
- F_Participant (PARAM_PARTICIPANT)
- F_QueueName (PARAM_QUEUE_NAME)
- F_RosterName (PARAM_ROSTER_NAME)

- F_SelectedResponses (PARAM_SELECTED_RESPONSE)
- F_StepDescription (PARAM_STEP_DESCRIPTION)
- F_StepName (PARAM_STEP_NAME)
- F_Subject (PARAM_SUBJECT)
- F_Tag (PARAM_TAG)
- F_WorkClassName (PARAM_WORK_CLASS_NAME)
- F_WorkflowName (PARAM_WORK_CLASS_NAME)
- F_WorkflowNumber (PARAM_WORKFLOW_NUMBER)
- F_WorkObjectName (PARAM_WORK_OBJECT_NAME)
- F_WorkObjectNumber (PARAM_WORK_OBJECT_NUMBER)

Refer to the *UI Toolkit APIs JavaDoc* reference documentation for more information on using these values with the appropriate controls.

How to Use This Guide

This guide uses specific navigation tools to help you find the appropriate content quickly: the **Contents**, the **Index**, and **Search** links.

- Click **Contents** to view the table of contents.
- Click **Index** to view the list of indexed information.
- Click **Search this document** to search for pages containing a specific word or term. (Click **Search tips** for help on using the search.)

Note If the Table of Contents does not load properly, click the **Text version of Contents** link in the upper right corner of the documentation window. Use the text version of Contents if you are using an older browser version, or you are experiencing problems viewing the default Table of Contents.

Conventions Used In This Guide

This guide uses the following type, text, and naming conventions:

Convention	Description
Bold type	Indicates menu names and items, file names, directory names, and notes, tips and warnings.
Italic type	Indicates referenced document titles and dialog box names.
Fixed width type	Indicates code samples, syntax, class names, and parameters. Green fixed width type indicates code sample comments. Typically, green fixed width type appears within code sample blocks.
Blue text	Indicates a link to another topic, a link to another section in the same topic, or a link to an external topic. In most browsers, when you pass the cursor over blue-colored text the text is underlined.
<drive>...	Indicates a directory structure that remains fixed relative to a dynamic location. For example, the directory structure for the HTML Step Processor files is **\FileNET\IDM\Web\IDMWS\Redist\WF_Step\html**; however, the directory structure could be located on any drive and contained in any directory on that drive.
Note	Includes information that you might find useful or want to know about in most situations.
Tip	Includes information you should treat as a general guideline while developing.

Caution	Includes information about practices, actions, or configurations that might cause data loss or compilation or application failures.
Product naming	Certain product names are abbreviated as follows: ● Designer, for eProcess Designer ● Administrator, for eProcess Administrator ● Tracker, for eProcess Tracker ● Configuration Console, for eProcess Configuration Console
Other Panagon eProcess documentation and terminology 	Access the eProcess documentation from any system that can connect to the eProcess web server by clicking the eProcess bar on the Panagon Web Services home page, then clicking the **Help** icon (shown to the left). For product terminology or definition of terms, refer to the *Glossary* in the eProcess documentation.

Printing Topics

Depending on need, you can print individual topics or the entire guide. Treat these instructions as guidelines for printing. (Depending on your printer type, the print menus or dialog boxes might display differently from those mentioned here.)

Printing topics

These instructions assume you are viewing and printing individual topics using a web browser (either Internet Explorer or Netscape). Refer to the online help for your web browser for more detailed information on printing from the browser window.

Internet Explorer

1. On the **File** menu, click **Print**.
2. Select the printing options you want. (The print options depend on the printer attached to the system.)

Note To print a frame or item in a frame, right-click the frame or item and click **Print**.

Netscape

1. On the **File** menu, click **Print**.
2. Select the printing options you want. (The print options depend on the printer attached to the system.)

Note The content of each frame prints separately by default.

Printing the guide

You must have Adobe Acrobat Reader 4.0 (or higher) installed on your system to print the entire guide.

From the Adobe Acrobat Reader, perform the following steps:

1. Open eprocess_dev_guide_5_0.pdf. (The file resides on the Web WorkFlo server, in the **<drive>...\Developer Files\Documentation\Developer Guide** directory in the Panagon eProcess Toolkit.)
2. On the **File** menu, click **Print**.
3. On the *Print* dialog box, locate **Print Range** and select **All**.
4. Click **OK**.

Note While all of the guide topics exist in the PDF version, not all of the navigation elements function in the same manner as in the online version. The PDF version is meant to allow you to print all topics in sequence.

Contact Us

Visit the FileNET web site

Visit the FileNET web site for information on the corporation, its products, Global Learning Services, Professional Services, customer applications, and user groups.

Visit the CSS web site

Go to the FileNET Customer Service and Support (CSS) web site for release notes, documentation, and other product-specific information.

Send documentation feedback

We welcome your comments on Panagon eProcess documentation. You may:

- Send email to the Director of Documentation.
- Call (714) 327-3449.
- Send a fax to (714) 327-5076.

License Agreement

READ CAREFULLY BEFORE INSTALLING. IF YOU DO NOT AGREE WITH THESE TERMS AND CONDITIONS, OR THE TERMS AND CONDITIONS IN YOUR SOFTWARE LICENSE AGREEMENT WITH FileNET, DO NOT INSTALL THE SOFTWARE. BY INSTALLING THE SOFTWARE, YOU AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE.

1 Authorization of End User

. Each Software product, including any documentation relating to or describing such Software, such as, but not limited to, logic manuals, flow charts and improvements or updates provided by FileNET (collectively called "Software"), is furnished to End User under a personal, non-exclusive, nontransferable license solely for End User's own internal use in compliance with this License and all applicable laws and regulations. End User agrees that this License does not permit End User to: (1) use the Software for a service bureau application or (2) transfer the Software without prior written consent from FileNET.

B. End User may make one copy of the Software, (with the proper inclusion of FileNET's copyright notice and any other proprietary notice and/or trademarks on such Software), for End User's own backup purposes; however, the End User may not otherwise copy or reproduce the Software except to install the Software on a single hard drive for use by End User in accordance with this License.

C. If End User is unable to operate the Software on the computer equipment due to an equipment malfunction, the Software may be transferred temporarily to other computer equipment during the period of equipment malfunction.

D. Title to and ownership of the Software and all of its parts (or any modifications, translations, or derivatives thereof, even if unauthorized) and all applicable rights in patents, copyrights and trade secrets in the Software shall remain exclusively with FileNET and its licensors. Software provided hereunder is valuable, proprietary, and unique, and End User agrees to be bound by and observe the proprietary nature thereof as provided herein. End User agrees to take diligent action to fulfill its obligations hereunder by instruction or agreement with its employees or agents (whose confidentiality obligations shall survive termination of employment or agency) who are permitted access to the licensed Software. Access shall only be given on a need-to-know basis. Except as set forth herein or as may be permitted in writing by FileNET, End User shall not use, provide or otherwise make available the Software or any part or copies thereof to any third party. Unless prohibited by law, End User shall not reverse engineer, decompile or disassemble the Software or any portion thereof, nor otherwise attempt to create or derive the source code. End User acknowledges that unauthorized reproduction, use, or disclosure of the Software or any part thereof is likely to cause irreparable injury to FileNET, who shall therefore be entitled to injunctive relief to enforce these restrictions, in addition to any other remedies available at law or in equity.

E. FileNET agrees that affiliates of End User may use the Software; provided that prior to any affiliate's use of the Software: (1) each affiliate shall sign and deliver to FileNET a mutually

agreeable appendix to this License certifying its License to be bound by the terms herein; and (2) such use by such affiliate shall be subject to the following: (i) End User accepts responsibility for the acts or omissions of such affiliates as if they were End User's acts or omissions; (ii) End User shall indemnify FileNET against losses or damages suffered by FileNET arising from breach of this License by any such affiliate as if effected by End User; and (iii) such use shall not constitute an unauthorized exportation of the Software or documentation under U.S. Government laws and regulations.

F. FileNET shall defend any action, suit or proceeding brought against End User so far as it is based on a claim that the use of any Software delivered hereunder constitutes an infringement of any patent, copyright or other intellectual property right, provided that FileNET is promptly notified by End User of the action and given full authority, information and assistance for the defense of the action. FileNET shall pay all damages and costs finally awarded against End User, but shall not be responsible for any compromise made without its consent. FileNET may at any time it is concerned over the possibility of such an infringement, at its option and expense, replace or modify the Software so that infringement will not exist, or request the End User to remove the Software involved and refund to End User an amount as established by FileNET.

G. FileNET shall have no liability to End User under any provisions of this Section, if any claim is based upon the use of the Software in combination with software not supplied by FileNET, or in a manner for which the Software was not designed, or if the claim of infringement would have been avoided but for End User's use of Software other than the latest unmodified release made available to End User.

2 Types of Software Licenses

. For purposes of this License, the following definitions shall apply:

1. Server Software is a copy of Software residing on a server or multiple servers.
2. Clients are personal computers, minicomputers, mainframes, workstations and terminal devices that interact with Server Software.
3. Client Software is a copy of Software residing on a Client that interacts with Server Software.
4. CPU is a central processing unit. A central processing unit can exist as a single instance or in multiple instances within a server.

B. End User agrees to license the Software as follows:

1. For Client Software - one copy of Client Software for each Client that accesses any Server Software.
2. For Server Software - one copy of Server Software for each server.
3. For CPU Software - one copy of Software for each CPU on a server containing FileNET Server Software. CPU licenses are categorized by performance as outlined in FileNET's CPU Classification Guideline. FileNET will not limit Enduser to a specific number of concurrent users under this licensing agreement. The number of CPU's deployed will be used as a means of calculating the overall system license price. Individual FileNET Systems cannot mix CPU Software licenses and FileNET's other Software pricing model, Concurrent User Licenses.

C. FileNET may audit the End User's Software usage remotely or on-site during service calls or upon reasonable notice during standard business hours. The audit shall determine Software usage by server for the number of concurrent Dedicated, Shared and/or eBusiness User Licenses required for each server. Should an audit determine that Customer's usage has exceeded the number of Concurrent User Licenses, or that End User's deployment of Software on Clients exceeds the contracted ratio of Clients to Concurrent User Licenses, End User agrees to purchase additional licenses in compliance with this Software Schedule. For a server with a combination of concurrent Dedicated, Shared and/or eBusiness User Licenses, any use beyond the total aggregate number of all Concurrent User Licenses (including previously acquired SLU type licenses) will require additional Concurrent User Licenses to be purchased. The highest ratio Concurrent User License identified on the End User's configuration and its applicable fee will be invoiced accordingly.

3 Termination

Notwithstanding the foregoing, FileNET shall have the right to terminate End User's authorization to use the Software if End User fails to comply with the terms and conditions of the License. Upon notice of such termination, End User shall immediately destroy the Software and all portions and copies thereof and, if requested by FileNET, shall certify in writing as to the destruction of the same.

4 Disclaimer and Limited Warranty

FileNET warrants the media on which the Software is furnished to be free from defects in materials and workmanship for a period of ninety (90) days from the date of purchase.

FileNET's entire liability and End User's exclusive remedy as to media shall be at FileNET's option, either the return of the amount paid for the Software or replacement of the media that does not meet FileNET's limited warranty and which is returned to FileNET with a copy of the receipt.

EXCEPT AS EXPRESSLY PROVIDED ABOVE, THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS", FILENET DOES NOT MAKE ANY AND HEREBY DISCLAIMS ALL WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The entire risk as to the quality, accuracy and performance of the Software and the Documentation is with End User. FileNET does not warrant that the functions contained in the Software will meet End User's requirements or that the operation of the Software will be uninterrupted or error free.

FILENET SHALL NOT BE LIABLE UNDER ANY CONTRACT, NEGLIGENCE, STRICT LIABILITY OR OTHER LEGAL OR EQUITABLE THEORY FOR ANY INDIRECT, CONSEQUENTIAL OR INCIDENTAL DAMAGES (INCLUDING DAMAGES FROM LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, PROCUREMENT OF SUBSTITUTE GOODS OR TECHNOLOGY, AND THE LIKE) ARISING OUT OF THIS LICENSE OR THE USE OF OR INABILITY TO USE THE SOFTWARE EVEN IF FILENET HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

5 Permitted Uses

This Software and the Documentation are licensed to you, the End User, and may not be transferred to or used by any third party for any length of time without the written consent of FileNET. You may not modify, adapt, translate, reverse engineer, decompile, disassemble, or create derivative works based on the Software or the written materials. Interface information necessary to achieve interoperability is available from FileNET on written request and payment of FileNET's then current fee.

6 Government Restricted Rights

The Software is commercial software and the Software and Documentation are provided with Restricted Rights. Use, duplication or disclosure by the Government is subject to restrictions as set forth in paragraph (c) (1) (ii) of the Rights in Technical Data and Computer Software Clause at DFARS 252.227-7013 or subparagraphs (c) (1) and (2) of the Commercial Computer Software Restricted Rights at 48 CRF 52.007-19, as applicable. Contractor/Manufacturer is FileNET Corporation, 3565 Harbor Blvd., Costa Mesa, California 92626.

7 Export

Regardless of any disclosure made by the End User or licensee to FileNET of the ultimate destination of the Software, End User shall not export or re-export, whether directly or indirectly, the Software to anyone outside the United States of America without first obtaining an export license from the United States Department of Commerce or any other agency or department of the United States Government as required.

8 Miscellaneous

This License is governed by the laws of California, without regard to its conflicts of laws, provisions, or the United Nations Convention on the International Sale of Goods and will be deemed a contract under seal. If any provision of this License shall be held by a court of competent jurisdiction to be contrary to law, that provision will be enforced to the maximum extent permissible, and the remaining provisions of this License will remain in full force and effect. This License is not assignable or transferable by End User without prior written consent of FileNET; any attempt to do so shall be void. Any notice, approval or consent required hereunder shall be in writing. The parties agree that a material breach of this License by End User would cause irreparable injury to FileNET for which monetary damages would not be an adequate remedy and that FileNET shall be entitled to equitable relief in addition to any other remedies it may have hereunder or at law. Section 4 shall survive any termination of this License.

Building and Deploying the Sample Java Processors

This topic describes the overall procedure used to build, deploy, and run an unmodified, working sample Step Processor or Launch Step Processor. These procedures step you through the process of compiling, installing, and deploying a sample Step or Launch Step Processor applet or application that is provided with the Panagon eProcess Toolkit. Understanding these procedures will assist you in customizing the sample applications or applets.

The overall procedure for building, compiling, deploying, and running a Step Processor or Launch Step Processor consists of a number of basic steps, each of which is itself a procedure or set of procedures. To help you avoid getting lost in the details of the overall process, the basic overall procedure is provided here, with links to the detailed procedure for each step.

1. Install the eProcess Toolkit to a new directory on your local system.
2. Install and configure the Java SDK.
3. Compile the Step Processor or Launch Step Processor samples.
4. Create a Java ARchive (JAR) file that contains the compiled versions of the samples.
5. Sign the JAR file.
6. Deploy the JAR file on the Web WorkFlo server.
7. Configure the server for the web application.
8. Run the Sample Step Processor application.

The remaining sections in this topic provide detailed instructions on accomplishing these steps.

Compile the Sample Processors

The Java samples provided with the Panagon eProcess Toolkit include both an application and an applet version of the Step and Launch Step Processors. If you have not already done so, you must perform the following actions before you can compile these samples:

- Install the Panagon eProcess Toolkit to a new directory on your local system.
- Install the Java SDK. (See the Configure the Java SDK topic for details on which JDK to use for your development environment and how to install it for eProcess. All of the procedures in this topic assume the JDK is installed in the default directory.)
- Configure the Java SDK to use the appropriate eProcess API class library.

Compile the unmodified sample Step and Launch Step Processors by performing the following steps:

1. Navigate to the local directory which contains the development files. If you kept the directory structure the same as the structure on the *Web WorkFlo and Toolkit* CD, the **<drive>...\Developer Files\vwpanel** directory on your development system contains the correct directories and files.

2. Select the **\samplestep** directory or the **\samplelaunch** directory, depending on which type of Java Processor you want to compile and test immediately. (Refer to the Java Step Processor Sample Files or the Java Launch Step Processor Sample Files topics for more information on the specific files available in each directory.)
3. Select all files and subdirectories in the selected directory and change the file attributes from Read-only to Read/Write.
4. Open a command prompt, and change to the directory containing the Step Processor sample files (for example, **\samplestep**).
5. Compile the unmodified Step Processor sample applet by entering a command similar to the following (modify for the appropriate JDK):

C:\jdk1.3.1\bin\javac.exe -classpath .;C:\<*your directory*>\pw.jar;%CLASSPATH% VWSampleStepApplet.java

Note You should specify the location of the package as part of the compiler command, as shown here. Alternatively, if you have already created a batch file that includes the **pw.jar** location in the CLASSPATH variable, you can specify `setenv` and need not specify the **pw.jar** location; for details, see the Configure the Java SDK topic).

Similarly, you compile the unmodified Launch Step Processor sample applet by entering a command similar to the following (from the directory containing the Launch Step Processor sample):

C:\jdk1.3.1\bin\javac.exe -classpath .;C:\<*your directory*>\pw.jar;%CLASSPATH% VWSampleLaunchApplet.java

Alternatively, if you want to use the sample Java applications, you can compile the unmodified Step Processor sample or Launch Step Processor sample applications by entering commands similar to the following (from the directory containing the appropriate sample):

C:\jdk1.3.1\bin\javac.exe -classpath .;C:\<*your directory*>\pw.jar;%CLASSPATH% VWSampleStepApplication.java

C:\jdk1.3.1\bin\javac.exe -classpath .;C:\<*your directory*>\pw.jar;%CLASSPATH% VWSampleLaunchApplication.java

6. Create a Java ARchive (.JAR) file containing the compiled classes.
7. Deploy the JAR file on the Web WorkFlo server

   If you compiled the applications only, or you only want to test the applications, you can run the sample Processor applications without first having to install them on the Web WorkFlo server.

Create a Java ARchive (JAR) file

Whenever you compile the unmodified applets samples, modify and rename the applets, or customize the samples, you must create a Java Archive file to contain the compiled classes. (Refer to Sun Microsystems JDK documentation for more information on using the Java Archive tool.) The following procedure assumes you have already compiled the samples without compiler errors.

Create a JAR file, for example for the Step Processor sample, by performing the following steps:

1. Open a command prompt, and `cd` to the parent directory of the directory containing the Step Processor sample (that is, **...\Developer Files**\).
2. Create a .JAR file containing only the compiled classes by entering a command similar to the following (modify for the appropriate JDK):

```
C:\jdk1.3.1\bin\jar.exe -cvf sample.jar samples\vwpanel\samplestep\*.class
samples\vwpanel\samplestep\resources\*.class samples\vwpanel\samplestep\images\*.class
samples\vwpanel\samplestep\images\*.gif
```

You should receive output similar to the following:

```
added manifest
adding: samplestep\VWSampleStepApplet.class(in = 2617) (out= 1297)(deflated 50%)
adding: samplestep\VWSampleStepApplication.class(in = 3090) (out= 1542)(deflated 50%)
adding: samplestep\VWSampleStepPanel.class(in = 4222) (out= 2053)(deflated 51%)
adding: samplestep\resources/VWResource.class(in = 437) (out= 299)(deflated 31%)
adding: samplestep\images/VWImageLoader.class(in = 1239) (out= 602)(deflated 51%)
adding: samplestep\images/icon.gif(in = 1432) (out= 1015)(deflated 29%)
adding: samplestep\images/step.gif(in = 1592) (out= 1260)(deflated 20%)
```

3. Once the classes are added to the JAR file, deploy the JAR file on the Web WorkFlo server.

Note You must sign the JAR file (the Netscape Object Signing Certificate is recommended).

If you intend to use several .JAR files, you should add all of the compiled files into a single .JAR file and deploy just that one file. Alternatively, you can specify multiple .JAR files using a comma (",") delimiter (however, be aware that not all browsers support multiple .JAR files). If you plan to use several JAR files, you should place each .JAR file in the same directory as the modified .ASP file that contains the applet.

Configure the Server for the Web Application

Before you can run a Step or Launch Step Processor application/applet (e.g., the sample Java Step Processor, `VWSampleStepApplication`, included with the Panagon eProcess Toolkit), if you have not already done so, you will need to use the Configuration Console to specify the web application defined for VWService. For details on how to configure the server for a web application, see "Managing VWServices" in the *eProcess Configuration Console Help*.

Once you have configured the server for the web application, you specify the web application when you run your application/applet. For example, to run the sample Java Step Processor, `VWSampleStepApplication`, you specify the Panagon web application as a parameter as follows: `webApp=WEB_WORKFLO` (previously, you would have specified: `product=PANAGON`). For information and examples on running an application/applet, see the next step, Run the Sample Step Processor Application.

Run the Sample Step Processor Application

If you compiled the Step Processor application samples, you will need to create a valid workflow on your WorkFlo Services server to run and test them. You can use Designer and Administrator to create a workflow and locate a valid work object number.

To run the unmodified sample Processor applications, use the following procedure:

1. In Designer, create a workflow.
2. Validate and run the workflow.
3. Retrieve a workflow object number (WobNumber) for an object in a queue from your workflow.
4. Run the Step Processor application sample by entering a command similar to the following:

   javaw -classpath .;<sample.jar>;C:\<*your directory*>\pw.jar;%CLASSPATH% <package name>.VWSampleStepApplication /hostName=<host name> /routerNames=<local router> /routerPorts=1099 /idmPort=1099 /webApp=WEB_WORKFLO /queueName=<queue name> /wobNumber=<object number>;

   where <sample.jar> is the .JAR file, the <package name> is the period-separated (".") directory structure in which the sample files reside, <host name> is the name of the Web WorkFlo server, <router> is the name of the router on the Web WorkFlo server, <queue name> is the name of the queue being accessed, and <object number> is the work object number (GUID) assigned to the work object, by WorkFlo Services.

   For example, you might enter a command similar to the following:

   javaw -classpath .;C:\Developer Files\samples\vwpanel\sample.jar;C:\<*your directory*>\pw.jar;%CLASSPATH% samples\vwpanel\samplestep\VWSampleStepApplication /hostName=asgard /routerNames=vwrouter /routerPorts=1099 /idmPort=1099 /webApp=WEB_WORKFLO /queueName=InBox /wobNumber=B8D270F79636E1489BA9EC2DD9F7E950
5. Run the Launch Step Processor application sample by entering a command similar to the following:

   javaw -classpath .;<sample.jar>;C:\<*your directory*>\pw.jar;%CLASSPATH%; <package name>\VWSampleLaunchApplication /hostName=<host name> /routerNames=<local router> /routerPorts=1099 /idmPort=1099 /webApp=WEB_WORKFLO /workflowDefinitionDocID=<>/mainAttachmentDocID=<> /subject=<subject>

   where <sample.jar> is the .JAR file, the <package name> is the period-separated (".") directory structure in which the sample files reside, <host name> is the name of the Web WorkFlo server, <router> is the name of the router on the Web WorkFlo or WorkFlo Services server, <queue name> is the name of the queue being accessed, <> is the workflow definition ID assigned to the workflow definition, <> is the library ID assigned to the attachment, and <subject> is the subject specified for the step that launches the workflow.

Panagon eProcess Services Architecture

Panagon eProcess Services consists of several components: Panagon Web Services, Panagon Web WorkFlo, and Panagon WorkFlo Services. This topic describes these components and how they interact with each other and with client applications in a typical, out-of-the-box (OOTB) configuration.

As illustrated in the figure below, when a user request does not require interaction with the Panagon WorkFlo Services through a Java applet or application, the request is routed through the Microsoft Internet Information Server (IIS) session to the Panagon Web Services and Panagon Web WorkFlo components. However, if the user request can be satisfied only through a Java applet, as is the case with Web WorkFlo applets, the applet residing on the client system gathers the necessary workflow information and communicates via RMI with the eProcess Router. The eProcess Router directs all RMI communication between the Panagon eProcess Services server and the server running Panagon WorkFlo Services.

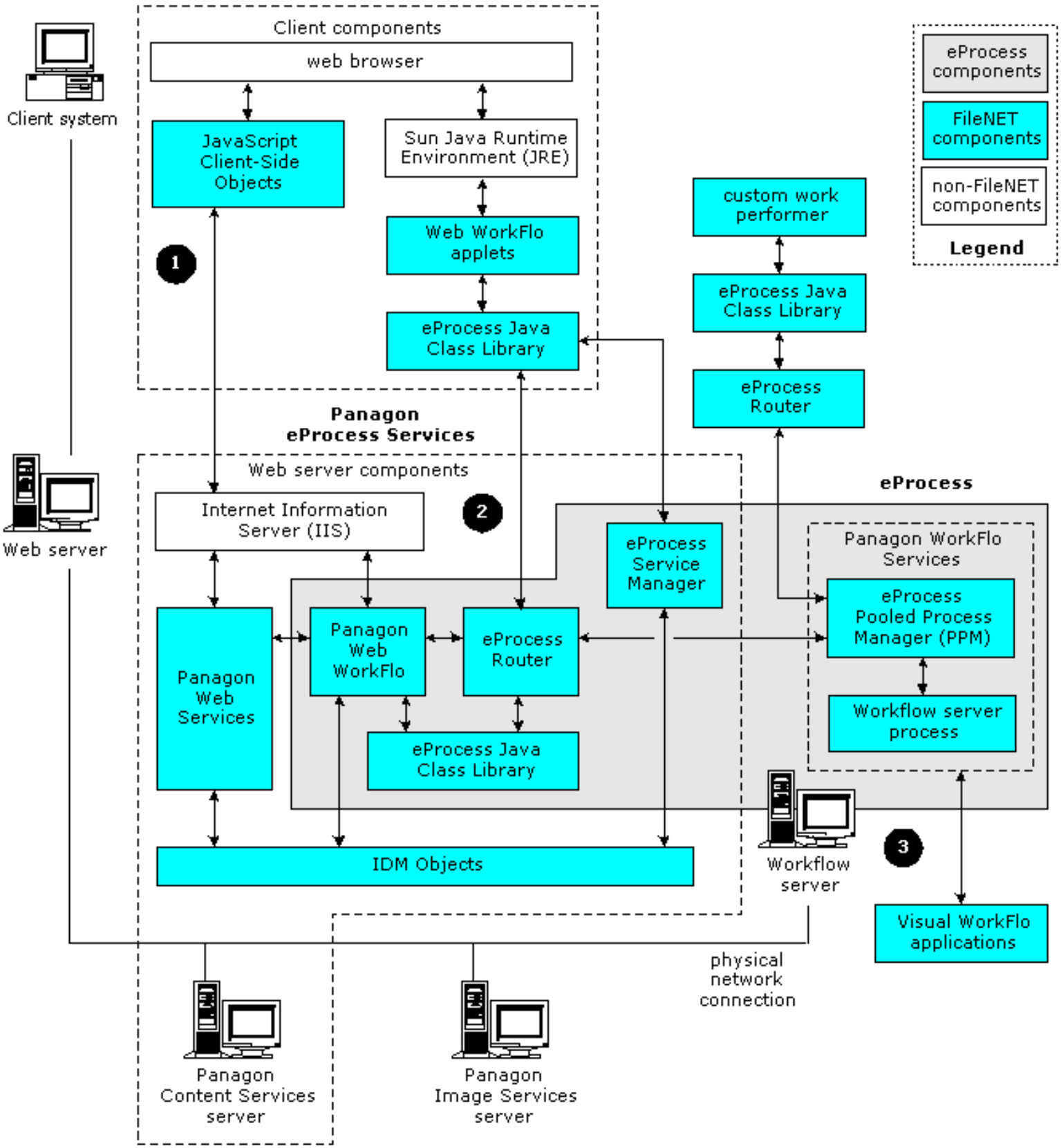
Note If the Web WorkFlo applet needs to interact with IDM Objects, as is the case with attachments, the applet communicates with the eProcess Service Manager.

Accessing the OOTB application in a web browser, a user logs into the Panagon eProcess Services server and either launches a workflow or participates in a running workflow. The out-of-the-box Web WorkFlo application requires Internet Explorer (Panagon Web Services and Web WorkFlo Services only support the Internet Explorer browser).

eProcess Services combines ASP and Java with the automation layer already provided by Panagon Web Services. The OOTB application contains Java applications and applets that use the Sun Java Runtime Environment (JRE) to operate. If a developer creates Java-based applications or applets, the JRE must be deployed on the client system as well as on the Web WorkFlo server system. If a developer creates only ASP-, HTML-, or XML-based processors (or workflow applications), the browser need not support Java and the JRE need not be deployed on the client system.

Note Panagon eProcess requires the JRE on the Panagon Web server, on the WorkFlo Services server, and also on the client (automatically downloaded when the client accesses the Web server for the first time). For information on which JDK to use and how to configure it for eProcess, see the Configure the JDK topic.

The following diagram illustrates where the eProcess Services architectural components reside in relation to each other and the client. (The figure illustrates only one common, generalized configuration. The figure is not intended to illustrate all possible configurations, and the relative size of any specific component does not signify the importance of the component.)



The numbers (below) refer to the corresponding area shown in the figure (above). The following table describes the architectural components and describes the constituent elements.

Architecture Area	Description

1 Client system	This section consists of a client workstation, with an installed web browser, connected to the network. Panagon eProcess Services automatically installs the Sun JRE and the eProcess Java class library (**pw.jar**) during the first client access to the Web server. However, you must manually install the necessary IDM COM controls the first time the client accesses the Web server. Once all of the necessary client-side components are loaded, the browser uses cached versions of the Web Services JavaScript objects for ASP communication to the Content Services or Image Services servers (as needed).
2 Panagon eProcess Services	This section consists of Panagon Web Services and Panagon Web WorkFlo on a web server, one (optional) Panagon Content Services (Windows NT 4.0 or Windows 2000) server, and an optional Panagon Images Services (on Windows NT, Windows 2000, AIX, Solaris, or HP-UX) server. In a typical eProcess Services configuration, the eProcess Router resides on the web server and acts as a Remote Method Invocation (RMI) routing mechanism for handling applet and application requests between the client system and the workflow server. (The eProcess APIs Java class library file, **pw.jar**, includes all of the eProcess APIs.) The eProcess Service Manager controls the Integration Service, which in turn facilitates communication (for IDM object-related operations) between the Web WorkFlo applets and the IDM Objects. All ASP related eProcess workflow requests are handled through the Panagon Web Services and Panagon Web WorkFlo applications. While not shown in the figure, eProcess Services ASP-related requests communicate with WorkFlo Services using the JiGlue COM Bridge. **Note** As indicated, the JRE runs on the Web WorkFlo Server and on the client. If you deploy custom applets based on the Java Step Processor or Launch Step Processors samples, the JRE must be deployed on all client systems.
3 eProcess	This section consists of WorkFlo Services (on Windows 2000, Solaris, HP-UX, or AIX platforms) and optionally, certain Panagon Image Services components (not shown), which are loaded during the Panagon WorkFlo Services installation. (A JRE must run on each WorkFlo Services server). The eProcess Pooled Process Manager (PPM) links the eProcess Router and the WorkFlo Services server processes. The eProcess PPM resides on the WorkFlo Services server, where it manages multiple threads of execution, in a sequential fashion, to a pool of WorkFlo Services server processes. When the PPM receives a request from the eProcess Router, the PPM forwards the request to the appropriate WorkFlo processes. In cases where the client machine is using either a Java- or COM-based workflow application (sometimes called a "work performer"), the client machine can either use an instance of the eProcess Router running on the client system to communicate to the WorkFlo Services server or the client can connect directly to the eProcess Router running on the workflow

server.

Note While not shown in the figure above, some Panagon Image Services components are installed on the WorkFlo Services server. Refer to the *Help for eProcess Administrators*, the *Panagon Web WorkFlo Installation Handbook*, and either the *Panagon WorkFlo Services for Windows Installation Handbook* or the *Panagon WorkFlo Services for UNIX Installation Handbook*, depending on the OS, for information on how to configure the Image Services components installed on the WorkFlo Services server.

Configure the Java SDK

Before you can develop Java applications or applets, or compile and run the out-of-the-box Java samples shipped with Panagon eProcess, you must first install and configure the appropriate Java SDK on your development system.

For the current eProcess release:

- For the Web client, use the Sun Microsystems JDK 1.4.0.
- For the Web server, use the Sun Microsystems JDK 1.3.1.
- For the WorkFlo Services server on a Windows 2000 platform, use both the Sun Microsystems JDK 1.3.0 and 1.3.1:
  - WorkFlo server: requires JRE 1.3.0.
  - NT Services: requires JRE 1.3.1.
- For WorkFlo Services on Solaris, use the Sun Microsystems JDK 1.4.0.
- For WorkFlo Services on HP-UX platforms, use the HP JDK 1.3.1.
- For WorkFlo Services on AIX, use the IBM AIX JDK 1.3.0 for AIX 4.3 and higher .
- For Panagon Open Client, the Sun JDK 1.3.1 is used on the web server, JDK 1.4.0 on the client (for information on developing eProcess Open Client applications, see the "Developing for eProcess Open Client" section in the *Panagon Open Client Developer's Guide*).

Note Versions used may be updated. For the latest information on which versions of software (including JDKs) are supported for Panagon eProcess Web Services, Web WorkFlo and WorkFlo Services, see the *eProcess Compatibility/Dependency/Server Matrix* document, located at http://www.css.filenet.com web site (go to **Product Info** > **Panagon** > **eProcess** > **Compatibility and Dependency**).

The Java sample applications supplied with the product are provided in the **pw.jar** file. However, there are known issues associated with adding **pw.jar** to your system CLASSPATH. These issues and the recommended Java SDK configuration procedure are described in this topic as follows:

Java SDK Configuration Procedure

On your development system, perform the following steps to configure your system to use the eProcess APIs:

1. Install the appropriate Java SDK on your development system (see above). Refer to the following web sites for information on downloading and installing the JDK:
   - For the Sun Microsystems JDKs for Windows and Solaris platforms, see http://java.sun.com/j2se/.
   - For the HP JDK, see http://www.hp.com/products1/unix/java/java2/sdkrte1_3/index.html.
   - For the IBM AIX JDK, see https://www6.software.ibm.com/dl/dka/dka-p.

   Note On Windows, the default installation directory is **<drive>...\jdk1.3.1**; on Solaris **<drive>:/j2sdk1.4.0** (for JDK 1.4.0), etc. (The remainder of this procedure assumes a Windows platform.) As a reminder, remember to add the JDK to your PATH variable. If you are planning to use Panagon eProcess to develop Java applications with the JiGlue COM Bridge, you must install the Java SDK in the default directory (for information on configuring the JiGlue COM Bridge on a development system, see Configure the COM Bridge).
2. Create a local directory (in any location) called **\process**. You will use this directory to hold the eProcess Developer files, including **pw.jar**.
3. Copy the **\Developer Files** directory (includes the **pw.jar** file) from the Panagon eProcess Toolkit CD to the new **\process** directory.

Refer to Install Panagon eProcess Toolkit for information on locating the **\Developer Files** directory on the CD (as well as additional information on its contents).

4. Use one of the following three methods to specify the location of the **pw.jar** file:

- Specify the **pw.jar** location each time you compile or run an application (not recommended).
- Create a batch file to include **pw.jar** in the CLASSPATH (recommended method if you are not working in an IDE).
- Use an IDE to add the **pw.jar** file location to your global or project setting (if using an IDE).

pw.jar CLASSPATH Issues

A known issue associated with adding the **pw.jar** file to the CLASSPATH is that this can cause security conflicts when attempting to run the out-of-the-box Java client applications on your development system. The conflict is caused when the Panagon eProcess Web WorkFlo server sends the correct **pw.jar**, but the CLASSPATH statement forces the client application to use the **pw.jar** file from the specified location.

If you run the Java Plug-in console on a system attempting to run the out-of-the-box Java client application with a **pw.jar** specified in the CLASSPATH, you will receive messages similar to the following:

```
    java.security.AccessControlException: access denied (java.lang.RuntimePermission
modifyThreadGroup)
    at java.security.AccessControlContext.checkPermission(Unknown Source)
    at java.security.AccessController.checkPermission(Unknown Source)
    at java.lang.SecurityManager.checkPermission(Unknown Source)
    . . .
```

There are three ways, any one of which will work, to get around this behavior:

- **Specify the pw.jar location:** Do not add the **pw.jar** file to the CLASSPATH. Instead, specify the correct location of the **pw.jar** file at the command line when you compile or run an application. For example, assuming the **pw.jar** file is located in the **c:\jdk1.3.1\process** directory, and you wanted to compile *MainSample.java*, you would **cd** to the directory containing the *MainSample.java* file and enter a command similar to the following:

  ```
  javac.exe -classpath .;c:\jdk1.3.1\process\pw.jar MainSample.java
  ```

  The disadvantage to this approach is that you have to specify the **pw.jar** location each time you compile or run the application.

- **Create a batch file:** Create a batch file that includes the **pw.jar** location in the CLASSPATH variable. This is the recommended approach, unless you are using an IDE (see next bulleted item). For example, your batch file should contain a statement similar to:

  ```
  set CLASSPATH=.;C:\<your directory>\pw.jar;%CLASSPATH%
  ```

  where:

  - You have installed the JDK in the default directory.
  - In this case, you are configuring the JDK on a Windows NT/2000 system.
  - You have not already set up the CLASSPATH for your JDK.

  Note The only portion of the CLASSPATH statement specific to eProcess development is the location of the **pw.jar** file. If any of these assumptions are not correct for your development environment, refer to the Sun Microsystems Java Development Kit documentation (http://java.sun.com/) for additional information on setting up the CLASSPATH variable.

- **Use an IDE:** Use an Integrated Development Environment (IDE) to compile and run your applications. If you are using an IDE (such as JBuilder or Visual Cafe), you need not modify the CLASSPATH. Instead of adding the **pw.jar** location to your system CLASSPATH, add the **pw.jar** file location to the specific project CLASSPATH or to the IDE global project setting. Refer to the documentation that came with your specific IDE for additional information.

Note If you are using an IDE, and you want to use the eProcess UI Toolkit Swing-based Java beans and interfaces to create user controls to allow interaction with the API samples, add the **vwpanel** to your project (the **vwpanel.jar** file is included with the eProcess Java UI Toolkit). For information on using the Process Java UI Toolkit, see the Java UI Toolkit Overview.

Required eProcess Services Components

Since much of Panagon Web WorkFlo operates within the Panagon Web Services architecture, when developing for eProcess Services you must reuse the contents of the following directories, which are part of the eProcess Services directory structure:

- **_ScriptLibrary**
- **\Application**
- **\Redist**
- **\WSAPI**
- **\WSCAPI**

The following directories are optional only if you are creating a new user interface and supply equivalent user interface components:

- **\CSS**
- **\Images**

Refer to the Panagon eProcess Services Architecture topic for more detailed information on the server configuration. Additionally, you should not modify the core files for the HTML Step processor or the Launch Step Processors. (Refer to the HTML Processor Toolkit Overview for more information.)

When developing for eProcess Services you will, in all likelihood, be adding new directories and files. You can reuse the modules, images, and style sheets available in the directories listed above. However, you must use all of the files in the root directory (as they were shipped), except for the following:

- **home.asp**
- **default.htm**
- **index.htm**
- **index.html**

You can modify the files listed above according to need. If you create your own web application, the start page must be at the same level as the **home.asp** file. The **default.htm**, **index.htm**, and **index.html** files all contain <META> redirection commands to **home.asp**. If you want to keep the **home.asp** and you have created a different top level page for your web application, then you must change the redirection command to reference your new page.

Note For more information on developing for eProcess Services, which includes the client-side, JavaScript-based API objects, refer to Locate the Panagon Web Services Toolkit Help for help in familiarizing yourself with the Panagon IDM toolkit.

HTML Processor Toolkit Overview

This topic provides an overview of the structure of the HTML Step Processor and Launch Step Processor toolkit and describes each of the components. Since these HTML processors are based on eProcess Services architecture, you cannot modify or extend these Step Processor or Launch Step Processor samples without first understanding how to access workflow sessions, objects, elements, queues, rosters, and logs using the the eProcess APIs. Additionally, you must understand how to use the Panagon Web Services JavaScript objects to access the Content Services and Images Services servers, and how the JiGlue COM Bridge functions.

This section contains information on the following topics:

- Modifying the default HTML Step Processor
- Modifying the default HTML Launch Step Processor
- Modifying the HTML interface files
- Modifying the HTML core components
- Modifying the HTML utility components

In addition, you will find reference topics on the following subjects:

- HTML processor toolkit descriptions and locations
- HTML Step Processor file relationships and dependencies
- HTML Launch Step Processor file relationships and dependencies

HTML processor operation

Before customizing the HTML processor files, you should know how the default HTML processors operate. Once the Step Processor is initialized, the web server supplies the step element property values. The values are stored on the client system, in the client-side objects. The property values stored on the client provide the means of updating the processor information. (Storing the property information on the client system reduces the number of server calls needed to complete the workflow step.) Once the processor step element is complete, the property values are converted to a XML string. Only the XML string is sent to the server. Once the server receives the XML string, the Web WorkFlo application parses the string and updates the values in the workflow.

The HTML Step Processors and Launch Step Processors use the JiGlue COM Bridge to access the attachments in the Content Services and Image Services libraries. When the HTML Step Processor or Launch Step Processor accesses attachments, the processors use the IDM Web Services Javascript objects; therefore, the user must supply valid user credentials to access the Content Services or Image Services libraries.

The HTML Step Processor will verify URL/UNC format when the user assigns an attachment. The processor expects a URL to begin with "http://". The processor expects a UNC to begin with "file://". The processor will warn if a local file is selected with a format of "file:/c:/" (where "c" is any letter

representing a drive).

Referenced Panagon Web Services Files

Most of the ASP files you create for eProcess Services will need to reference some, or all, of the Panagon Web Services Javascript library and ASP object implementation files. This topic describes these files and provides example lines of code to reference the files.

Topics include:

Essential Panagon Web Services Files

This table lists the essential Panagon Web Services files and briefly describes each file.

Referenced file	Description
InitApplication.asp	Reference this file in top level pages only. This file implements session-scoped variables and constants needed by Panagon Web Services. In addition, this file sets cookie states.
InitPage.js	Reference this file in each page in your application. This file implements page-scoped variables and constants.
EnableRemoteObject.asp	Enables remote scripting by calling the remote scripting components in the _ScriptLibrary directory (in the root web structure). Remote scripting must be enabled to make method calls on the client-side Javascript API objects, which are used in both the Web Services and the Web WorkFlo application pages.
PopupManager.js	Provides functions for handling popup windows.
Misc.js	Provides the ability to use the Web Services utility functions.
Document.js	Modified for Panagon WorkFlo. Provides the ability to create an IDMWSC_Document object and use methods supported by the object.
Event.js	Provides the ability to create an IDMWSC_Event object and propagate events.
Folder.js	Provides the ability to create an IDMWSC_Folder object and use methods supported by the object.

Library.js	Modified for Panagon WorkFlo. Provides the ability to create an IDMWSC_Library object and use methods supported by the object.
Result.js	Provides the ability to create an IDMWSC_Result object and use the methods and access properties supported by the object.
StoredSearch.js	Provides the ability to create an IDMWSC_StoredSearch object and use the methods supported by the object.
Version.js	Provides the ability to create an IDMWSC_Version object and use the methods supported by the object.

Note All of the referenced files are located in the \IDMWS\WSCAPI directory on the eProcess Services web server. For more information on how to use these referenced Web Services files or how to use the Javascript API objects, refer to Locate the Panagon Web Services Toolkit Help.

Example Code to Reference Files

Include the following lines in the approximate order shown. Include the line referencing the InitApplication.asp in the top level page only.

```
<script language="Javascript" src="WSCAPI/InitApplication.asp"></script>
<script language="Javascript" src="WSCAPI/InitPage.js"></script>
```

The lines referencing the Javascript objects can be included in any order, but you should list them in the order the object is most likely to be used. In general, you should include only the .JS file associated with the object you will create in a given page.

```
<script language="Javascript" src="Redist/PopupManager/PopupManager.js"></script>
<script language="Javascript" src="WSCAPI/Library.js"></script>
<script language="Javascript" src="WSCAPI/Folder.js"></script>
<script language="Javascript" src="WSCAPI/Document.js"></script>
<script language="Javascript" src="WSCAPI/Event.js"></script>
<script language="Javascript" src="WSCAPI/Result.js"></script>
<script language="Javascript" src="WSCAPI/Misc.js"></script>
```

During web application operation, the referenced Javascript library files are cached on the client system, so including all of the .JS files in every application page impacts performance only during the first page request. After the files are cached in local cache, all calls to the .JS files are handled by the locally cached copies.

You must use remote scripting to access the client-side API objects. You must include the following line to enable remote scripting:

<!-- #INCLUDE FILE="../WSCAPI/EnableRemoteObject.asp"-->

Referenced Panagon Web WorkFlo Files

Many of the ASP files you create for eProcess Services might need to use these referenced Panagon Web WorkFlo application files. This table lists most of the essential Panagon Web WorkFlo files and briefly describes each file (or file set).

Referenced file	Description
WF_Extras\Attachment.js	Provides client-side functionality for viewing an attachment in its native application or in the IDM Viewer. Called by Attachment.asp. Both the Attachment.js and the Attachment.asp files use the Panagon Web Services Javascript objects extensively. Refer to Referenced Panagon Web Services files for general information about these object. Refer to Locate the Panagon Web Services Toolkit Help for information on where to locate detailed information.
WF_Extras\Attachment.asp	Implements the server-side functionality for viewing attachments.
WF_Extras\globalInfo.asp	Provides eProcess router and Content Services port numbers. Additionally, this file specifies the FileNET-supplied and custom .JAR file locations needed for both default and customized Java applets.
WF_Extras\verifySession.asp	Provides the ability to verify the session information. Used in HTML Step Processors and Launch Step Processors to determine the base URL information. If the base URL value is NULL, the function implemented in the file works with PWRedirector.asp to set the correct base URL (window.location.href) information.
WF_Extras\Token.js	Provides client-side functionality for retrieving the session token for the current session.
WF_Logon\isPWLogon.asp	Provides the ability to get VWSession information for Step Processor and Launch Step Processors currently attached to the Web WorkFlo server. If no session exists, this file provides the ability for the user to be prompted to log on to the appropriate session.

PWRedirector.asp	Provides a means of retrieving the session information from Panagon Web Services for the current Web Services session. The PWRedirector.asp file resides in the root directory, and redirects remote calls to the proper pages in the Web WorkFlo application. If the user does not start from the root web directory of the Web Services server, many of the application-scoped variables used in Web WorkFlo are either unavailable or NULL. This requirement is especially important for mail notifications referencing Step Processors and Launch Step Processors.
WF_Launcher\open_launcher.js	Must be included in any file that launches a Launch Step Processor. See workflow_main.asp for usage.
WF_Step\open_step.js	Must be included in any file that launches a Step Processor. See workflow_main.asp for usage.
WF_FileOpen\FileOpen.js WF_FileOpen\OtherSelect.js	Provides client-side functionality for opening and working with attachment files, which are stored in Content Services or Image Services libraries, from the Step Processor toolbar. These files use the Panagon Web Services Javascript objects extensively. Any file referencing these files must handle events associated with the IDMWSC_Event object.
WF_FileOpen\FileOpen.asp WF_FileOpen\OtherSelect.asp	Implements the server-side functionality for opening and working with documents from the Step Processor toolbar. These files are called by the corresponding .JS files on the client.

Configure the COM Bridge

This topic describes how to configure the JiGlue COM Bridge on a development system. Configuring the JiGlue COM Bridge consists of the following basic steps:

- Configure JDK, CLASSPATH, and register the jiglue.dll
- Deploy the client application

The remainder of this topic provides procedures on how to perform each of these steps.

Configure JDK, CLASSPATH, and Register the jiglue.dll

To use the JiGlue COM Bridge, if you have not already done so, you must first install the appropriate Java SDK on your development system (the JDK version varies depending upon your OS). For information on which JDK to use and how to configure a JDK on a development system for Panagon eProcess, see the Configure the Java SDK topic. Once you have installed the JDK, you must specify the location of the **pw.jar** file (contains the eProcess APIs), and register the **jiglue.dll** file on your system.

If you are planning to develop COM applications on your Web WorkFlo server, you have typically already installed your JDK, configured your CLASSPATH, and the **jiglue.dll** was registered during eProcess installation; skip to the Deploy the Client Application procedure.

If you are planning to develop an application on a system that is neither your workflow server or Web WorkFlo server, use the following procedure:

1. If you have not already done so, install the appropriate Java SDK on your development system. In order to use the JiGlue COM Bridge, you should install the JDK in the default directory (on Windows, the default installation directory is **<drive>...\jdk1.3.1**; on Solaris **<drive>:/j2sdk1.4.0** (for JDK 1.4.0), etc.). For information on installing the JDK and configuring your system for eProcess development, see the Configure the Java SDK topic.
2. Create a local directory (in any location) called **\process**. You will use this directory to hold the eProcess Developer files, including **pw.jar** and the JiGlue files.
3. Copy the **jiglue.dll**, **jiglue.tlb**, and **pw.jar** files to the **\process** directory. In order for JiGlue to work correctly, *all three of these files must be in the same directory* (as they are as shipped on the Panagon eProcess Toolkit CD). Refer to the Panagon eProcess Toolkit topic for information on these files and where to find them.
4. If you have not already done so on this system, configure the CLASSPATH to specify the **pw.jar** location (typically either by creating a batch file to include **pw.jar** in the CLASSPATH, or if you are developing using an IDE (such as Visual Cafe, JBuilder, Forte, etc.), by adding **pw.jar** to your global or project setting). For details, see the Configure the Java SDK topic.

   Note The **jiglue.dll** parses the CLASSPATH for the location of the **pw.jar** file. If JiGlue does not find **pw.jar** in the CLASSPATH:

- The JiGlue library attempts to find the registry key: HLM\Software\FileNET\Panagon eProcess Services - Web WorkFlo\Extras.
- If the library cannot find this registry key, **jiglue.dll** uses the default **<drive>...\WF_Extras** path, appends **pw.jar** to it, and adds the location to the CLASSPATH for the running JVM process.
- If **jiglue.dll** does not find any registry entry, the library determines the path to **jiglue.dll** and assumes the **pw.jar** is located in the same directory. JiGlue appends **pw.jar** to its own location in the path and adds the location to the CLASSPATH for that instance of the JVM.

5. Open a command prompt, and `cd` to the \process subdirectory.
6. Register jiglue.dll by entering the following at the command prompt:

   ```
   regsvr32.exe jiglue.dll
   ```

   Tip You can unregister the library by entering the following command:

   ```
   regsvr32.exe /u jiglue.dll
   ```

Deploy the Client Application

If you are developing on the Web WorkFlo server, you do not need to register the **jiglue.dll** and you do not need to install the JRE. Both steps were done during installation.

If you are developing a new ASP-based application or extending the current ASP application, the appropriate JiGlue COM Bridge configuration already exists on the eProcess Web WorkFlo server. However, if you are developing stand-alone workflow applications (which are sometimes referred to as work performers) and you plan to deploy them on client systems, your installer must perform the following steps:

1. Install or check for an existing installation of the JRE (currently for the client, JRE 1.4.0). Both **JRE\bin** and **JRE\bin\client** must be in the system path.
2. Create a new directory to contain the process components.
3. Copy the **jiglue.dll**, **jiglue.tlb**, and **pw.jar** files to the new directory (all three of these files must be in the same directory).
4. Register the **jiglue.dll** (see Step 6 above).

Start a Local Router (for development)

This topic describes how to start a local eProcess router on a development system. An eProcess router is a Java Remote Method Invocation (RMI) communication bridge. You need not manually start a local router if an eProcess Router is running on either the Web WorkFlo server or the WorkFlo Services root server. In those cases, you can pass the host name, port number, and router name as part of the `VWSession.logon() router_URL` parameter. However, starting a local router enables you to observe the RMI communication between a development system and a workflow server more closely.

This procedure assumes you have already installed and configured the Java SDK to use the eProcess APIs. This procedure also assumes you are using the appropriate JDK (in the example shown, the Sun Microsystems JDK 1.3.1 on a Windows platform; for details on which JDK version to use and how to configure it for eProcess development, see Configure the Java SDK).

The following procedure uses the Panagon Task Manager to start a local router (treat all entries as case-sensitive):

1. Open a command prompt and enter the following command to start the Task Manager (assumes for this example that **pw.jar** is in the *process* directory; modify for the JDK as appropriate):

C:\jdk1.3.1\bin\javaw.exe -classpath C:\process\pw.jar filenet.vw.apps.taskman.VWTaskApplication

2. When the *Task Manager* window displays, click **Start**.
3. In the router dialog box, enter a name for the local instance of the router; for example, `myrouter`.
4. Enter the name of the server running Web WorkFlo Services. For example, if the name of the server running WorkFlo services is "hq-workflo", enter `hq-workflo`.
5. Enter a valid WorkFlo Services user name. (When running the samples, you should use an WorkFlo Services administrative user.)
6. Enter the password for the specified user.
7. Enter an isolated region number for the isolated region you are using for development. (This number can be from 1 to 999 but the number should correspond to an initialized isolated region. Although it is possible to do so, you should not run the samples (or other eProcess applications in development) in an isolated region already used for either production or other development.
8. Click **OK**.

Once the message appears stating that the local router has started, you can use it to communicate with a workflow server.

Tip On Windows 98/NT/2000 systems, you can create a short-cut to launch the eProcess Task Manager. In the short-cut properties dialog box, enter the command shown in Step 1 (above) in the target field. Specify the directory containing the API samples as the start directory. (Specifying the directory containing the API samples, creates and updates the router trace files in that directory, which can be useful for debugging when you start modifying the sample code.)

The first time you attempt to connect to a session, a **vwapi.txt** file will be created in the root directory of

the disk on which the application is being run. The file contains a complete list of all session transactions handled by the router. If you don't start the router, the **vwapi.txt** file continues to grow rapidly (as duplicate error messages are appended to it) until you terminate the sample application.

Using the COM Bridge

This topic provides samples of how to use the JiGlue COM Bridge to create a VWSession object and access content from an Image Services or Content Services library, as follows:

- Using the JiGlue COM Bridge to create a VWSession object
- Using the JiGlue COM Bridge to access an Image Services or Content Services Library

Note The JiGlue COM Bridge requires that elements of a Variant array be of a proper type, and they must be of the same type. For details on data types and using arrays, see the JiGlue COM Bridge Data Types topic.

Using the JiGlue COM Bridge to create a VWSession object

The following sample syntax demonstrates how to dimension and create an instances of JiGlue and VWSession objects in both Visual Basic and VBScript.

Note How you use the JiGlue COM Bridge to create a VWSession object depends on the development environment.

Visual Basic syntax:

```
dim objJiGlue As New JiGlue.JiGlueUtil
dim objVWSession As Object
Set objVWSession = objJiGlue.newInstance("filenet.vw.api.VWSession")
```

VBScript syntax:

```
dim objJiGlue, objsession
set objJiGlue = CreateObject("JiGlue.Util")
set objsession = objJiGlue.newInstance("filenet.vw.api.VWSession")
```

Once you create a VWSession object, you can access any of the methods or property in the eProcess APIs. For detailed reference information on the APIs, refer to the *eProcess API JavaDoc* documentation. The *eProcess API* documentation is HTML-formatted class information taken directly from the Java™ code (using the Sun Microsystems Javadoc utility). Refer to the Install Panagon eProcess Toolkit topic for more information on viewing the *eProcess API JavaDoc* documentation.

Using the JiGlue COM Bridge to Access an Image Services or Content Services Library

Important As of the current Panagon eProcess release, eProcess can, if desired, be set up without Content Services library access. For information on configuring eProcess without Content Services, refer to the *WorkFlo Services Installation Handbook.*

When accessing the content stored in either Image Services or Content Services libraries, you must first

connect to the libraries using IDM Objects.

As an example, if you wanted to access attachments from a Content Services library or an Image Services library from an ASP file, you might include code similar to the following (the following code sample was taken from the sample **launchStepElementInit.asp** page and modified to make it easier to follow. For additional information, refer to the **launchStepElementInit.asp** file on the Web WorkFlo server):

```
Dim objLib, objDoc, vwAttachment, parts, objJiGlue
```

'Log on to the CS or IS library using the IDM Objects.

```
set objLib = Server.CreateObject("IDMObjects.Library")
objLib.LogonID = IDMWS_Library_GetLogonID(libraryName)
set objDoc = objLib.GetObject(Application("idmObjTypeDocument"),ObjectID)
docLabel = objDoc.Label
docVerNum = objDoc.Version.Number
```

'Create a eProcess VWAttachment object using the information retrieved
'from the CS or IS library.

```
set objJiGlue = CreateObject("JiGlue.Util")
set vwAttachment = objJiGlue.newInstance("filenet.vw.api.VWAttachment")
parts = Split(DocID, ":")
if (UBound(parts) >= 0) then
 vwAttachment.setType(CInt(parts(0)))
end if
if (UBound(parts) >= 1) then
  vwAttachment.setLibraryType(CInt(parts(1)))
end if
if (UBound(parts) >= 2) then
  vwAttachment.setLibraryName(parts(2))
end if
if (UBound(parts) >= 3) then
  vwAttachment.setId(parts(3))
end if
if (UBound(parts) >= 4) then
  Dim id
  id = parts(3) + ":" + parts(4)
  vwAttachment.setId(id)
  vwAttachment.setVersion(parts(4))
end if
```

JiGlue.Util Reference

This section describes the method supported by the JiGlue.Util object:

- newInstance

Tip To use the JiGlue COM Bridge in Visual Basic, add it to your current project. To use the JiGlue COM Bridge in VBScript, in a non-Web WorkFlo application, register the file on the web server.

newInstance()

The method creates an COM object that represents the specified Java object. In most cases, you need only to create an instance of the VWSession object to be able to access other objects in the eProcess APIs. This is the most often used method in the Web WorkFlo application.

Note The Jiglue.newInstance method can only instantiate an object whose class has a default constructor (that is, without any argument).

Syntax

newInstance(className)

Parameters

- className: String. Specifies the name of the object to create. The string must contain the package location of the Java class being instantiated. For example, if you want to create an instance of the VWAttachment class in an ASP application, you would have to specify a command similar to the following:

set vwAttachment = JiGlue.newInstance("filenet.vw.api.VWAttachment")

Visual Basic example:

```
Dim objJiGlue As New JiGlue.JiGlueUtil
Dim objAttachment As Object
Set objAttachment = objJiGlue.newInstance("filenet.vw.api.Attachment")
```

VBScript example:

```
dim objJiGlue, objAttachment
set objJiGlue = CreateObject("JiGlue.Util")
set objAttachment = objJiGlue.newInstance("filenet.vw.api.Attachment")
```

Returns

A COM object representing the Java class specified in the className parameter.

Back to top.

JiGlue COM Bridge Data Types

This topic describes JiGlue COM Bridge data types, including:

- COM Bridge data type conversions
- COM Bridge data types guidelines

COM Bridge Data Type Conversions

The JiGlue COM Bridge supports the following Java to COM data type conversions:

Java data type	COM data type
boolean (java.lang.Boolean)	Boolean (VT_I1)
char (java.lang.Char)	2-byte unsigned (VT_UI2)
integer (java.lang.Integer)	4-byte signed integer (VT_I4)
short (java.lang.Short)	2-byte signed integer (VT_I2)
long (java.lang.Long)	2-byte signed integer (VT_I4)
float (java.lang.Float)	Float (VT_R4)
double (java.lang.Double)	Double (VT_R8)
java.lang.String	BSTR (VT_BSTR)
java.util.Date	Date (VT_DATE)
Other Java object	IDispatch (VT_DISPATCH)

COM Bridge Data Types Guidelines

- For simple (non-object) types, you need not use the Set statement in Visual Basic or VBScript code, as illustrated in the following general, sample statements:

```
boolean = queuelement.getFieldValue("booleanField");
integer = queuelement.getFieldValue("intField");
float = queuelement.getFieldValue("floatField");
string = queuelement.getFieldValue("stringField");
time = queuelement.getFieldValue("timeField");
```

  However, if a Java method returns an object other than the java.lang.* (see below), the `Set` statement is needed.
- JiGlue supports both one-dimensional and multi-dimensional arrays (previously JiGlue only supported one-dimensional arrays). The following API methods (in addition to the new Attribute-related methods — see What's New in 5.0) that take multi-dimensional arrays as parameters can be called for JiGlue:

```
VWEventDefinition
     String[][] getAssignments()
     void setAssignments(String[][] theAssignments)


VWCompoundStepDefinition
```

```
        VWInstructionDefinition createCreateInstruction(String theWorkClassName,
    String[][] theFieldAssignList)
        VWInstructionDEfinition createAssignInstruction(String[][] assignPairs)


    VWStepDefinition
        String[][] getPreAssignments()
        void setPreAssigtnments(String[][] thePreAssignments)
        String[][] getPostAssignments()
        void setPostAssignments(String[][] thePostAssignments)
```

- When specifying an integer array, be aware that in Visual Basic, an integer is a 16-bit value, whereas in Java (and eProcess), an integer is a 32-bit value. If, for example, you specify a 2-byte integer array in Visual Basic, it might be mapped in JiGlue to java.lang.Short, thereby causing the API to throw an exception. For example, the following call to setDataFields will fail:

  ```
  Dim integerArray(2)

  integerArray(0)=10
  integerArray(1)=20
  integerArray(2)=30

  wob.setFieldValue "integerArrayField", integerArray, false
  dataFields = wob.getDataFields(127,1)
  wob.setDataFields dataFields, false
  ```

  However, if you change the array to 4-byte integers, as follows, the call to setDataFields succeeds:

  ```
  Dim integerArray(2)

  integerArray(0)=CLng(10)
  integerArray(1)=CLng(20)
  integerArray(2)=CLng(30)

  wob.setFieldValue "integerArrayField", integerArray, false
  dataFields = wob.getDataFields(127,1)
  wob.setDataFields dataFields, false
  ```

- The JiGlue COM Bridge allows you to remove explicit assignments for the object references. The following examples demonstrate two ways of using the data types supported by the JiGlue COM Bridge:

  Example 1:

  ```
  Dim myDate As Variant
  Dim mySimpleDate As Object
  Set mySimpleDate = JiGlue.newInstance("java.text.SimpleDateFormat")
  myDate = mySimpleDate.parse("09/12/2001 11:58 am")
  ```

  Example 2:

  ```
  Dim myDate As Variant
  Dim mySimpleDate As Variant
  Set mySimpleDate = JiGlue.newInstance("java.text.SimpleDateFormat")
  myDate = mySimpleDate.parse("09/12/2001 11:59 am")
  ```

  In the examples shown above, mySimpleDate was declared as both Object and Variant (in the different examples); in both cases, the object reference was assigned by using Set. Notice that in both cases myDate was declared as Variant without using the Set statement. The reason this works as coded, is because SimpleDateFormat.parse returns java.util.Date which is mapped to Visual Basic's Date type (which is non-object — see table above). As previously indicated, if a Java method returns an object other than the java.lang.* (as indicated above), the Set

statement is needed. Note that the Jiglue.newInstance method can only instantiate an object whose class has a default constructor (that is, without any argument).

Add JiGlue COM Bridge to Visual Basic

To use the JiGlue COM Bridge in Visual Basic, add the JiGlue.dll file to your project as a reference.

Follow these general steps to add JiGlue to your project:

1. From within the Visual Basic project window, select the **Project** > **References**.
2. From the *References* dialog box, click **Browse**.
3. Navigate to the directory containing the **JiGlue.dll** file.
4. Select **JiGlue.dll**. Click **Open**.
5. In the *References* dialog box, scroll down the *Available References* list, and check box next to *JiGlue*.

Your Visual Basic application can now access the JiGlue.JiGlueUtil class in the library.

Use JiGlue COM Bridge in ASP

This sample demonstrates how to use the JiGlue COM Bridge to access queue information in an ASP-based application. The primary advantage of creating an ASP-based solution for working with the JiGlue COM Bridge is that the JiGlue COM Bridge need not be registered and deployed on each client system. In contrast, if you develop a Visual Basic application, you must configure and register the JiGlue COM Bridge on the client system the application can communicate with WorkFlo Services.

This topic includes:

Creating a Sample VWSession object (using JiGlue) to Query Queues

The sample creates a VWSession object and uses that object to query queues for information.

```
<%@ LANGUAGE="VBSCRIPT" %>
<HTML>
<title>List Queues using the JiGlue COM Bridge</title>
<body>
<%

'Dimension the necessary variables. In this example, the web_server_name
'and router_name variables are assigned explicit, literal values.
'An alternate method might be to pass the information in from another page using
'the Request.QueryString(), which is part of the VB Object Model.
'Refer to the appropriate Microsoft documentation for more information.

Dim web_server_name, router_name, objJiGlue, session
Dim queues, description
web_server_name = "asgard"
router_name = "vwrouter"

'Create an instance of JiGlue, and use it to create a VWSession object.

set objJiGlue = CreateObject("JiGlue.Util")
set session = objJiGlue.newInstance("filenet.vw.api.VWSession")

'Call the logon() method on the newly created session object.
'The sample statement shown here passes SysAdmin as both the username and
'password and constructs the RMI statement using the
'values specified for the web_server_name and router_name variables.

session.logon "SysAdmin","SysAdmin","rmi://" & web_server_name & "/" & router_name
```

```
%>

<!-- Start a table list the queues that are available on the system. -->

<table border="0" width="80%" bgcolor="#EEEEEE">
<tr><td colspan="2" bgcolor="#CCFFCC"><strong>Available Queues</strong></td></tr>
<tr><td bgcolor="#CCFFCC"><strong>Name</strong></td>
<td bgcolor="#CCFFCC"><strong>Type</strong></td></tr>
<%

'Retrieve an array of queue names using the newly-created session object. This example
'passes an integer value of 23 to retrieve all user, process (or work), and
'system queues. The integer value is the result of a logical OR of the values associated
'with each queue type. Refer to the VWSession.fetchQueueNames() method description,
'in the eProcess API reference documentation, for more information.

Dim queueValue
queues = session.fetchQueueNames(23)
for i=0 to UBound(queues)

'Using the array of queue names, retrieve the VWQueue object for
'each queue name found. Query the VWQueue object to determine the queue type.

set objQueue = session.getQueue(CStr(queues(i)))
queueValue = objQueue.getQueueType
%>

<!-- Start a row, and create a cell with the current queue name. -->

<tr><td width="33%">
<%=queues(i)%>
</td>

<!-- Create a cell with the containing the description of the queue type based on the integer value
returned from the getQueueType() method. -->

<td width="66%">
<%

'Convert the retrieved value to a human readable description of the queue type.

if queueValue = 1 then
description = "Work"
elseif queueValue = 2 then
description = "User"
else
description = "System"
end if
%>
<%=description%>
```

```
</td></tr>
<%
Next
%>

<!-- Close the table once all queues have been retrieved. -->

</table>
<%

'Call the logoff() method on the VWSession object to end the session and log off.

session.logoff
%>

<P>This sample HTML and VBScript code demonstrates one way of creating a VWSession object, logging on to the server, retrieving a list of available queues, displaying the retrieved queue information, and logging off the server.</P>
</body>
</html>
```

Running the Sample ASP

You can run this sample by copying the contents of the code block (above) to an empty file called **queue.asp**. Place the **queue.asp** file in the **<drive>...\Program Files\FileNET\IDM\Web\IDMWS** directory on your Web WorkFlo server, and enter the address for the explicit page.

For example, if the Web WorkFlo server is named Asgard, you would enter an address in a web browser similar to the following:

http://asgard/idmws/queue.asp

For the code shown above to work you must have the following:

- At least one server running WorkFlo Services.
- An eProcess Services server properly configured and running a eProcess Router (configured to communicate with the Workflow Services),
- At least one least one initialized isolated region within WorkFlo Services.
- At least one valid user name and password for a WorkFlo Services user.

Tip Another method of establishing a WorkFlo session for an .ASP based application is by using the **global.asa** file to set application-scoped variables and objects (like the web server name or the router name). The application-scoped variable approach allows you to develop event handlers that run automatically when special ASP events occur and instantiate well-known IDM objects.

Active Server Pages Overview

Some of the Panagon Web WorkFlo and most of the Panagon Web Services components rely on the ASP object model and server-side scripting to function. This topic provides a high-level overview of ASP, as it is used in eProcess Services.

Server-side scripts run when a browser requests an .asp file from the web server. The server processes the ASP file from top to bottom and executes the script commands in the sequence they appear in the page. Because the scripts run on the server, the web server performs all of the necessary processing and returns the requested data back to the client browser in a HTML page – without the server-side scripting code.

You can include client-side script in the same ASP page. The server-side script executes before the server returns a response, but the client-side script is sent as part of the response. Additionally, you can include other pages in your ASP. The included files are treated as part of the ASP file during execution. The included files are executed from beginning to end before returning to the page that called the file.

There are two benefits to using server-side scripting:

- The Web Server, not the client system, processes the request. Normally, this behavior reduces the number of round trips for a typical request.
- The client cannot view the source of server-side scripting, so your development effort is less likely to be compromised. Once the server completes processing the page, the user sees only the generated HTML as well as non-HTML content, such as XML, on the pages being viewed.

Any statements placed between the <%. . .%> delimiters are treated as script statements. While you can use any script language supported by the Microsoft Script Host, the Panagon Web Services and Web WorkFlo components use VBScript for all server-side scripting and Javascript for all client-side scripting.

ASP Objects

You can use the following objects in the Active Server Page Object Model: Server, Application, Session, Request, and Response. The Server object is actually the operating environment for the Active Server Pages. (Since the WorkFlo architecture does not rely on MTS transactions, you should not attempt to use the ObjectContext object.)

For more information on the Active Server Page Object Model objects, their methods and properties, and how to use them, refer to the Microsoft MSDN documentation.

ASP Example

This example illustrates the structure of a simple ASP file, what the client's browser might display, and shows the source received by the client for the request.

Assume that an ASP file contains the following content:

<%@ Language=VBScript %>

```
<HTML>
<HEAD>
<TITLE>Sample ASP Page showing a loop</TITLE>
</HEAD>
<BODY>
<H1>Looping a variable</H1>

<%
' Before sending the response back to the client, loop
' through a count 5 times and use a Response.Write
' statement to print the current loop count in the HTML document.

Dim loopNum, loopString

' Dimension two variables; one for holding an integer and one
' for holding the dynamically updated, HTML formatted string.

For loopNum = 1 to 5
  loopString = "<font color='#0000FF'>The current loopNum count equals: " & loopNum &
"</font><BR>"
Response.Write(loopString)
Next
%>

<P>This paragraph is static HTML content.</P>

<SCRIPT Language="Javascript">
/* This is an example of client-side code executed on the client. */
document.write("Today's date is " + Date());
</SCRIPT>
</BODY>
</HTML>
```

When a client requests the ASP file containing the source shown above, the user sees content similar (the formatting might be different) to the following in the client browser:

Looping a variable

The current loopNum count equals: 1
The current loopNum count equals: 2
The current loopNum count equals: 3
The current loopNum count equals: 4
The current loopNum count equals: 5

This paragraph is static HTML content.

Today's date is Fri Sep 12 06:00:00 2036

If the user views the source, delivered as a response the initial ASP request, he or she would see the

following HTML formatted code:

```
<HTML>
<HEAD>
<TITLE>Sample ASP Page showing a loop</TITLE>
</HEAD>
<BODY>
<H1>Looping a variable</H1>
<font color='#0000FF'>The current loopNum count equals: 1</font><BR>
<font color='#0000FF'>The current loopNum count equals: 2</font><BR>
<font color='#0000FF'>The current loopNum count equals: 3</font><BR>
<font color='#0000FF'>The current loopNum count equals: 4</font><BR>
<font color='#0000FF'>The current loopNum count equals: 5</font><BR>
<P>This paragraph is static HTML content.</P>
SCRIPT Language="Javascript">
/* This is an example of client-side code executed on the client. */
document.write("Today's date is " + Date());
</SCRIPT>
</BODY>
</HTML>
```

Note While the Javascript source included in the original ASP file was included in the response sent to the client browse, none of the server-side VBScript statements were included.

Run the Unmodified Samples

To ensure that your development system is set up correctly and to gain a basic understanding of how to connect to a session, query a workflow queue, and access work objects, it is recommended that you first compile and run the API samples as delivered (without modifying the samples' Java source code).

To run the API samples, perform the following steps:

1. Compile the API samples
2. Run the sample application
3. Review the sample output

Note To run the API samples you must have workflow server running an active eProcess Pooled Process Manager (PPM). To view additional information about the eProcess Pooled Process Manager, see Panagon eProcess Services architecture. The samples will not run without being able to attach to the Web WorkFlo Server.

Compile the API samples

1. If you have not already done so, configure the JDK on your development system.
2. `cd` to the directory containing the API samples (including the main sample file, *MainSample.java*), and enter a command similar to (assumes the Sun JDK 1.3.1 on a Windows platform; modify as needed):

   ```
   C:\jdk1.3.1\bin\javac.exe -classpath .;C:\<your
   directory>\pw.jar;%CLASSPATH% MainSample.java
   ```

   Alternatively, you need not specify the location of **pw.jar** if you have already created a batch file (to set variables) that includes the **pw.jar** location in the CLASSPATH variable (for details, see the Configure the Java SDK topic). If this is the case, you could specify:

   ```
   setenv
   C:\jdk1.3.1\bin\javac.exe MainSample.java
   ```

   Or you can compile all of the classes in the directory at once. For example:

   ```
   setenv
   C:\jdk1.3.1\bin\javac.exe *.java
   ```

Note The complete pathname for **javac.exe** is not necessary if your system PATH variable already includes the corresponding **javac.exe**.

Run the sample application

Once you have compiled the samples, you can run them. The command line syntax for the running the MainSample application is as follows:

MainSample <username> <password> <server name>:<port number>/<router instance name> [<wfDefinition_filename> | <wfDefinition_filename> <output_filename>]

where: <username> is an existing user name in the eProcess Service (for example, SysAdmin); <password> is the valid password for the specified user; <server name> (defaults to the local machine) and <port number> (defaults to the default port number) are entities on the web server, and <router instance name> is the name you want to specify for your local instance of the router object.

The first three parameters are required; the remaining parameters are optional. The <wfDefinition_filename> is an existing workflow definition (.pep) file, and the <output_filename> is the file name where you want the results of the application directed.

Suppose, for example, you want to run the unmodified samples as SysAdmin, with a router instance on server Asgard, port 1099, named `myrouter`, without specifying workflow definition (*.pep) or output files. In this case you would enter a command similar to the following:

java -classpath .;%CLASSPATH% MainSample SysAdmin Asgard:1099/myrouter

When the MainSample application runs without encountering errors, you will receive output similar to the following, in the same command window from which you ran the MainSample application:

Writing messages to file: MainSample.out
~~ Starting the main sample.

~ Starting SysConfigSample execution.
~ SysConfigSample execution complete.

~ Starting WFDefinitionSample execution.
~ WFDefinitionSample execution complete.

~ Starting LaunchSample execution.
~ LaunchSample execution complete.

~ Starting RosterSample execution.
~ RosterSample execution complete.

~ Starting StepProcessorSample execution.
~ StepProcessorSample execution complete.

~ Starting QueueSample execution.
~ QueueSample execution complete.

~ Starting MilestoneSample execution.
~ MilestoneSample execution complete.

```
~ Starting OperationsSample execution.
~ OperationsSample execution complete.

~ Starting SystemStepSample execution.
~ Launching Generated Workflow
~ SystemStepSample execution complete.

~ Starting StepProcessorSample execution.
~ StepProcessorSample execution complete.

~ Starting ProcessSample execution.
~ ProcessSample execution complete.

~ Starting StepProcessorSample execution.
~ StepProcessorSample execution complete.

~ Starting LogSample execution.
~ LogSample execution complete.

~~ MainSample execution complete.
```

Note This output represents the typical console output results for the first time the MainSample application is run. If the MainSample application is run a second time, without modification, you will receive notification stating the required queues already exist.

Review the sample output

You will notice that several files appeared as result of running the samples. This section lists and describes, in general terms, each of the files created by the unmodified samples.

- **MainSample.out**: This is the default file name for the output file. You can specify a different output file name as part of the MainSample application command line. The output file contains the text-formatted output of the MainSample application. The results of the application operation in the output file clearly indicates where and in what order the other classes are called. Additionally, the contents of the output file provide some indication of the type of information you can retrieve from each associated API.
- **Sample.pep**: This is the default file name of the workflow definition created by the MainSample application. You can specify a different workflow definition file, if you have a valid workflow definition file already. This workflow definition file contains the XML representation of the workflow definition. You can open this text-formatted file to examine the workflow definition elements, or you can open the file in Designer and look at the graphical representation of the workflow.
- **vwapi.txt**: This file contains all of the session related information. The file logs all of the session-related calls; the information contained in the file is useful to resolve problems connecting to and making calls to an eProcess Server.

Note If you followed the procedures for compiling and running the samples, the output files should reside in the same directory as the *MainSample.class* file.

Step Processor and Launch Step Processor Overview

This section contains information describing the structure and operation of both the HTML Step and Launch Step Processors and the Java Step and Launch Step Processors samples. Additionally, this section includes general information on customizing the existing samples.

Within the context of Panagon Web WorkFlo and WorkFlo Services, the terms "Step Processor" and "Launch Step Processor" have specific meanings, as follows:

- **Step Processor:** This applet or application processes the information and resources a workflow participant needs to complete a step in a workflow. When a workflow participant opens a work item, the Step Processor displays the instructions, Content Services or Image Services attachments, current field values, response options, or other resources necessary to allow the participant to process the step. When designing or customizing a Step Processor, you must account for data and information gathered prior to calling the Step Processor. In other words, a typical Step Processor will receive information from another, prior Step Processor.
- **Launch Step Processor:** This is a more specialized type of Step Processor. Launch Step Processors begin or launch the workflow. Typical Launch Step Processors contain all of the information necessary to initialize the workflow. Launch Step Processors are independent of the condition of the workflow progress, so you need consider only what information the launch step will introduce into the workflow.

The Step Processor and Launch Step Processor samples included in the Panagon eProcess Toolkit represent a close approximation of the Processors as they are implemented in Panagon Web WorkFlo. (In the case of the HTML processors, the toolkit files are the actual Step Processor and Launch Step Processor modules, components, and files implemented in Web WorkFlo.) While you can reuse these samples in a web-based application, these Processor samples are not meant to be the only starting place for workflow development. Rather, these samples represent only one way of developing applications that can launch workflows or process steps in a workflow.

To modify the Step Processors or Launch Step Processors you must first know how to use the eProcess APIs. In fact, one helpful way of looking at Step Processors and Launch Step Processors is to view them as specialized applications that make extensive use of the eProcess APIs. The sections in this guide discussing how to modify the HTML- and Java-based versions of the samples assume that you have some understanding of the eProcess APIs and how they are used.

Panagon Web WorkFlo includes two sets of Toolkit files, depending on the development focus, for understanding and modifying Step Processors and Launch Step Processors:

- **HTML Processors**: the HTML Step Processor and Launch Step Processor Toolkit files are installed on the Panagon Web WorkFlo server during installation.
- **Java Processors**: the Java Step Processor or Launch Step Processor Toolkit files are included on

the *Panagon Web WorkFlo and Toolkit CD*. (Refer to Install Panagon eProcess Toolkit for additional information.

Note The Java and HTML Step Processors support a Read-only mode. This mode opens when an end user only has Read-only permission on a work queue in which the work object is located. The user is notified of the Read-only mode when the Step Processor opens. However, in Read-only mode any changes a user makes are discarded. Additionally, while in the Read-only mode the Step Processor interface displays only the following options: Cancel, Status (if the workflow definition allows), and Help.

Java Step Processor Samples Overview

The Java Step Processor and Launch Step Processor use the UI Toolkit APIs to initialize and display user interface components and controls; however, these samples make extensive use of the eProcess APIs. You cannot modify or extend these Step Processor or Launch Step Processor samples without first understanding how to access workflow sessions, objects, elements, queues, rosters, and logs using the eProcess APIs.

The samples are meant to illustrate how to extend the existing Step Processor and Launch Step Processor applets and applications using the reusable interface components. The samples are not meant to be used as a framework for developing universally adaptable custom applications to launch workflows or process workflow steps.

The Java processor toolkit allows you to use the supplied UI (User Interface) Toolkit components in one of two ways: (1) use a text-based editor to program directly using the UI Toolkit components, or (2) use an Integrated Development Environment (IDE) for drag-and-drop RAD (Rapid Application Development). Either development option requires that you understand AWT and Swing. The runtime versions of the UI Toolkit classes are included in the **pw.jar** file; however, if you are using an IDE for RAD configure your IDE to use the **vwpanel.jar** file. (The **vwpanel.jar** contains the necessary classes and images for RAD programming.)

This section includes topics that provide information on the following topics:

- Compiling and running the supplied Step Processor samples (without modification)
- Installing (deploying) the custom Step Processor (or Launch Step Processor) on the Web WorkFlo server
- Customizing and compiling the Step Processor (or Launch Step Processor) samples with slight modifications

In addition, this section includes reference and development information on the following topics:

- Java Step Processor sample file descriptions
- Java Launch Step Processor sample file descriptions
- Java UI Toolkit class, interface, and bean descriptions
- Java UI Toolkit parameters supported by specific controls
- How to work with attachments using these controls

HTML Processor Toolkit Files

Most of the HTML Processor Toolkit files are included in the \Redist\WF_Html_Toolkit directory (on the eProcess Services server). The directory includes the following sub-directories:

- \Core: includes the essential .ASP, .INC (server-side includes), and .JS (Javascript script) files used in the default Step Processor and Launch Step Processor. All the files in this sub-directory are used for defining objects, initializing processor state, and enabling workflow processing.
- \Images: includes images (GIF format) used in the default Step Processor and Launch Step Processor. This topic does not list the contents of the \WF_Html_toolkit\Images directory. You can add image files to the \Images directory, but do not remove any of the existing image files.
- \UI: includes .ASP files used to format the default Step Processor and Launch Step Processor user interface. The files contained in this sub-directory define the step and Launch Step Processor user interface behavior.
- \Utility: includes client-side (.JS) and server-side (.ASP) files for calling utility functions for quick sorting, property type checking, and XML streaming.

The tables in this section list and describe the files in the \Redist\WF_Html_Toolkit directory. The following sections describe the contents of each of the sub-directories under the **\WF_Html_Toolkit** directory:

Core Shared Files

The following table lists, and briefly describes, the shared files in the \WF_Html_Toolkit\Core directory. You can extend these files, but do not delete the files or any of the existing content of these files.

Note For more information, see the Modify Core Files topic.

File name	Description
att.js	Implements functions for creating a client-side attachment object and defines the associated methods for the object. Checks for the attachment type: document, folder, and stored search. Uses some of the Web Services client-side, Javascript API objects to work with attachments.

launchStepElementInit.asp	Retrieves the launch step information and initializes the Launch Step Processor system field values for the user interface components. References work groups and attachment functionality. Uses loadAtt.inc, loadFields.inc, and loadWorkGroup.inc to initialize the attachment properties.
launchStepProcessor.asp	Implements functions for parsing and processing XML string for launch processor operations (like launch and cancel). The implemented functions assume that an VWSession already exists.
loadAtt.inc	Server-side include file. Implements general functions for loading and displaying attachment objects (including attachment properties).
loadFields.inc	Server-side include file. Implements general functions for loading, displaying, and sorting system data fields arrays.
loadWorkGroup.inc	Server-side include file. Implements general functions for loading and displaying work groups arrays.
step.js	Implements functions for creating and working with client-side work group objects, step element objects, step information objects, and step parameter objects. The functions define all of the associated methods for these objects.
stepElementInit.asp	Retrieves the step information and initializes the Step Processor system field values for the user interface components (like F_StepDescription, F_Originator, F_LaunchDate, etc.). References work groups and attachment functionality. Uses loadAtt.inc, loadFields.inc, and loadWorkGroup.inc to initialize the attachment properties.
stepProcessor.asp	Implements functions for parsing and processing XML string from Step Processor operations (like save, cancel, reassign, return, etc.). The implemented functions assume that an VWSession object already exists.
stepProcessorAttHelper.asp	Implements helper functions for getting and saving attachments.
stepProcessorFieldHelper.asp	Implements helper functions for working getting field values and working with field arrays.
stepProcessorWGHelper.asp	Implements helper function for saving workgroup information.
toolbarHelper.asp	Implements helper functions for both Step Processor and Launch Step Processor user interface toolbar components (like save, cancel, move, cancel launch, etc.)

User Interface Files

The following table lists, and briefly describes, the shared files in the \WF_Html_Toolkit\UI directory. You can extend these files, but do not delete the files or any of the existing content of these files.

Note For more information, see the Modify User Interfaces Files topic.

File name	Description
arrayEdit.asp	Implements functions for checking, displaying, and editing field arrays. The embedded form posts the information to the dataField.asp file.
attachment.asp	Container file. Implements a frameset for presenting the attToolbar.asp and attachmentTable.asp files.
attachmentTable.asp	Provides a table for displaying attachment. The file implements Javascript functions for getting attachment names, item information, attachment IDs, and the name of the library storing the attachment.
attToolbar.asp	Implements functions for creating an attachment toolbar and working with the changing toolbar state.
dataField.asp	Receives array data from the arrayEdit.asp file. Provides a table for editing non-array values and displaying field array information.
generalInfo.asp	Provides a table to display general step information, like the subject or comments. Implements functions used in the General tab.
milestone.asp	Provides support for displaying milestone-related message and status information.
reassign.asp	Implements a dialog box that allows a participant or tracker to reassign the workflow step to a new participant.
resource.asp	Defines localizeable strings that are shared by the HTML Step Processor, HTML Launch Step Processor, and HTML milestone tracker components.
stepInfo.asp	Displays general step information (like the subject, sender information, origination time, reception time, comments, and responses).
tab.asp	Implements functions for selecting the Attachments, Data Fields, Workflow Groups, General, and Milestone tabs.

tabContent.asp	Container file. Implements logic for selecting the page(s) associated with the selected tab.
toolbar.asp	Works with toolbarHelper.asp, which implements all of the button event handlers, and ToolbarButton.asp files to implement functions to display a toolbar for the following operations: Complete, Save, Move, Cancel, Reassign, Return, Status, and Help.
workGroup.asp	Provides a table to display workgroup information, like workgroup name and workgroup participants. Implements function for calling workGroupEdit.asp.
workGroupEdit.asp	Provides a dialog box for displaying and updating workgroup information.

Utility Files

The following table lists, and briefly describes, the shared files in the \WF_Html_Toolkit\Utility directory. You can extend these files, but do not delete the files or any of the existing content of these files.

Note For more information, see the Modify Utility Files topic.

File name	Description
utility.js	Client-side file that implements Javascript utility functions for working with field values; Defines functions for sorting, checking, and converting strings to XML.
utility.asp	Server-side .ASP file that implements VBScript utility functions for working with field values; Defines functions for sorting, checking, and converting strings from XML to HTML.

Toolbar Shared Files

The following table lists, and briefly describes, the shared toolbar files in the \Redist\Toolbar directory. These files are supplied by Panagon Web Services.

File name	Description
ToolbarButton.asp	Implements toolbar icon functionality. Supplied by Panagon Web Services but referenced to by the HTML processors user interface components, like attToolbar.asp.

ToolbarStoredSearch.asp	Implements toolbar icon functionality for the stored searches. Supplied by Panagon Web Services but referenced to by the HTML processors user interface components.

Java Launch Step Processor Sample Files

The Java Launch Step Processor sample files are located in the **\Developer Files\Samples\vwpanel\samplelaunch** directory. Refer to Install Panagon eProcess Toolkit for information on installing the appropriate development files on your system.

The directory contains the following files:

File name (and location)	Description
VWSampleLaunchApplet.java	For applets hosted in .ASP files. Implements methods for loading the window by using the VWResource and VWImageLoader classes. Creates and initializes the processor application. Loads the window, sets the title, establishes a VWSession, and calls the main user interface panel.
VWSampleLaunchApplication.java	For applications. Implements methods for loading the window by using the VWResource and VWImageLoader classes. Creates and initializes the processor application. Loads the window, sets the title, establishes a VWSession, and calls the main user interface panel.
VWSampleLaunchPanel.java	Implements the user interface visual elements for the Launch Step Processor. Initializes the controls for the panel.
\resources\VWResource.java	Acts as a resource provider for the processor. The default resource is the title string for the applet.
\images\VWImageLoader.java	Loads images for the VWSampleLaunchPanel container.
\images\icon.gif	Image file for VWSampleLaunchPanel container.
\images\launcher.gif	Image file for VWSampleLaunchPanel container indicating that this is a Launch Step Processor.

Java Step and Launch Step Processor Toolkit Overview

This section provides an overview of the resources provided as part of the Java Step and Launch Step Processor Toolkit that may be used to develop a customized Java Step or Launch Step Processor.

Topics covered in this overview include:

- Java Step and Launch Step Processor Toolkit
- Processor Session Sharing

Java Step and Launch Step Processor Toolkit

The Java Step and Launch Step Processor Toolkit includes of the Java UI Toolkit APIs. These are Java Interfaces and Beans that can be used to customize the look and feel of the user interface you create for a Java Step or Launch Step Processor. You may use these Interfaces and Beans instead of, in addition to, or in conjunction with Swing or AWT components to create the user interface. In other words, you can mix and match components to suit the interface.

The Java processor toolkit allows you to use the supplied UI (User Interface) Toolkit components in one of two ways: (1) use a text-based editor to program directly using the UI Toolkit components, or (2) use an Integrated Development Environment (IDE) for drag-and-drop RAD (Rapid Application Development). The toolkit also provides a VWPanel container to hold the UI Toolkit components. You may add the **vwpanel.jar** as a panel in your preferred IDE (such as Visual Cafe, JBuilder, Forte, etc.) for easy drag and drop building of the interface.

The runtime versions of the UI Toolkit classes are included in the **pw.jar** file. If you are using an IDE, configure your IDE to use the **vwpanel.jar** file. (The **vwpanel.jar** contains the necessary classes and images for RAD programming.)

The Java Step Processor and Launch Step Processor use the UI Toolkit APIs to initialize and display user interface components and controls, as illustrated by the Step Processor samples provided with your installation. Note that these samples make extensive use of the eProcess APIs. You cannot modify or extend these Step Processor or Launch Step Processor samples without first understanding how to access workflow sessions, objects, elements, queues, rosters, and logs using the eProcess APIs.

- Compiling and running the supplied Step Processor samples (without modification)
- Installing (deploying) the custom Step Processor (or Launch Step Processor) on the Web WorkFlo server
- Customizing and compiling the Step Processor (or Launch Step Processor) samples with slight modifications

In addition, this section includes reference and development information on the following topics:

- Java Step Processor sample file descriptions

- Java Launch Step Processor sample file descriptions
- Java UI Toolkit class, interface, and bean descriptions
- Java UI Toolkit parameters supported by specific controls
- How to work with attachments using these controls

Processor Session Sharing

The Java Step Processors shipped with Web WorkFlo run as applets hosted by the Java Plug-in in an ASP page, or as a dialog window that is opened by another window. One significant difference between the operation of the Java and HTML versions of the Step Processor and the Launch Step Processor is session sharing behavior. If the Java Step Processor opens from another eProcess window, they share the logon session, and the user need not logon again.

By contrast, the HTML Step Processors and Launch Step Processors are built on top of the JiGlue COM Bridge and the IDM Web Services Javascript objects. Therefore, any steps requesting attachments require the user to supply separate user credentials. If a Java Step Processor is opened from an email notification, it is run as an applet and behaves the same way as the HTML Processors.

Deploying a Custom Java Step or Launch Processor

Once you have created a new Java Step Processor or Launch Step Processor and have added the classes to a Java ARchive (JAR) file, you must deploy and configure the custom Processor on the eProcess server.

Note Your JAR file must be signed (with either a Netscape Object Signing Certificate or a Sun Java Object Signing Certificate). For additional information, see Sign a Jar File.

Caution If you upgrade to a new Panagon Web Services version, you must redeploy all existing custom Step and Launch Processors. Existing custom Java Step Processors and Launch Step Processors should be recompiled and then redeployed using the procedure shown here. If you get compiling errors, correct your samples by comparing them to the new version's sample Java Processors (see Java Step Processor Sample Files).

To deploy a custom Java Step or Launch processor, use the following procedure:

1. On your Web WorkFlo server, navigate to the <drive>...**Program Files\FileNET\IDM\Web\IDMWS** directory.
2. Create a new directory under the **\Redist\WF_Step** directory (at the same level as the **\Tabbed** directory). For Launch Step Processors, create the directory under **\Redist\WF_Launcher**.
3. Copy the **step_main.asp** file from the **\Tabbed** sub-directory, and put the copy into your newly created directory. For Launch Step Processors, copy the **launcher_main.asp** file.
4. Rename the .ASP file (it is recommended that this name reflect the function of the custom processor). Change the file attribute from Read-only to Read/Write.
5. Open the renamed file. Locate the <PARAM> attribute of the <OBJECT> tag that matches the following line:

<PARAM NAME=CODE VALUE="filenet.vw.apps.steps.tabbed.VWTabbedStepApplet">

6. Modify the <PARAM> statement to reference the correct applet. For example, if your JAR file was created in the **\newStep** directory and the applet class is called newStepApplet, your parameter statement would be similar to the following:

<PARAM NAME=CODE VALUE="newstep.newStepApplet">

7. Save the renamed file.
8. Navigate to the **\Redist\WF_Extras** directory.
9. Place your custom Processor Java archive (JAR) file in the **\WF_Extras** directory.
10. Still in the **\WF_Extras** directory, locate the **globalInfo.asp** file. Change the file attribute from Read-only to Read/Write.
11. Open the **globalInfo.asp** file, and locate the following commented statement:

's_jarArchiveTag=s_jarArchiveTag + ", " + "samplestep.jar"'

12. Copy the statement, and paste a new instance of the line immediately below the first. Do not use or modify the statement that specifies the location for the **pw.jar** file.
13. Remove the apostrophe (" ' ") from the beginning of the line to allow the server to parse the statement. (By default, the line is commented out using the VBScript comment convention.)
14. Delete the "samplestep.jar" portion of the statement, and modify the statement to reflect the actual name and location of the JAR file you created and copied to the **\WF_Extras** directory.

For example, the statement should be similar to the following:

s_jarArchiveTag=s_jarArchiveTag + ", " + "<JAR file>.jar"

(where **<JAR file>.jar** is the name of the JAR file you created and copied to the **\WF_Extra** directory).

Note: You can also, if you wish, place your jar file in another directory. However, since the jar file path is relative to "idmws\redist\wf_extras", you must specify the path. If you do this and use unified logon, the folder must have anonymous access enabled. Remember, as previously indicated, the jar file must be signed (for details, see Sign a Jar File).

15. Save the modified **globalInfo.asp** file.
16. Add the processor to the workflow.

If you intend to use several .JAR files, you should add all of your class files into a single JAR file and reference only the one file. Alternatively, you can specify multiple JAR files using a comma (",") delimiter. Not all browsers support multiple JAR files.

Customize the Step Processor Sample

This topic describes how to customize (or extend) the Java Step Processor sample applet. Refer to the Java Step Sample Files topic for a detailed list of the available sample files. These procedures assume that you have already:

- Compiled and tested the unmodified processor samples successfully.
- Worked with and understand the eProcess Java APIs.
- Have some familiarity with Swing controls and containers (for information, see the Java UI Toolkit Overview topic).

Customizing the Step Processor sample consists of two key steps:

1. Create a new version of the sample Step Processor
2. Customize the code (for example, by adding additional user controls)

Note These procedures assume you are using the Java SDK in native, command line mode (see Configure the Java SDK for details on which JDK to use for your development environment and how to install it for eProcess) and that the JDK is installed in the default directory. If you are using a Java Integrated Development Environment (IDE), such as Visual Cafe, JBuilder, Forte, etc., you can use the eProcess Java UI Toolkit API Rapid Application Development (RAD) components, in a drag-and-drop fashion, by adding the vwpanel.jar file to the Java project (for additional information, see the Java UI Toolkit Overview topic). The eProcess Java UI Toolkit provides ready-to-use user controls to simplify customizing the applet user interface. Refer to the documentation that came with your IDE for information on adding drag-and drop RAD components.

Create a new version of the sample Step Processor

These procedures demonstrate how to modify and compile customized versions of the Step Processor samples. However, you can adapt these procedures to update the Launch Step Processor samples by changing the directory and file names used in these procedures.

1. Navigate to the **\vwpanel** directory, copy the directory and rename the **\samplestep** directory (typically to reflect the name of your customization goal). For example, **\newstep**.

Caution You should rename the directory to ensure that package names do not conflict when you create the Java ARchive (JAR) file (see step 7 below).

2. Change to the renamed directory.
3. Rename the following sample files, by adding a prefix, to match your customization. For example, you might rename the files as shown:
    - Rename VWSampleStepApplet.java to **newStepApplet.java**
    - Rename VWSampleStepApplication.java to **newStepApplication.java**
    - Rename VWSampleStepPanel.java to **newStepPanel.java**
4. Open each renamed file, and replace every occurrence of samples.vwpanel.samplestep with the

name of your new directory, which reflects the package structure. For example, if your new directory is called **newstep**, replace samples.vwpanel.samplestep with samples.vwpanel.newstep.

5. Open each renamed file, and replace every occurrence of **VWSample** with the prefix you added to the file name. For example, in the example shown above, you would replace VWSample with new. All class definitions must match the files names.
6. Compile the samples by entering a command similar to the following (modify for the appropriate JDK):

   C:\jdk1.3.1\bin\javac.exe -classpath .;C:\<*your directory*>\pw.jar;%CLASSPATH% newStepApplet.java

   Alternatively, you can compile all of the classes in the directory at once (assuming, for example, that you have already created a batch file that includes the **pw.jar** location in the CLASSPATH variable; for details, see the Configure the Java SDK topic), as follows:

   setenv
   C:\jdk1.3.1\bin\javac.exe -classpath .;C:\<*your directory*>\pw.jar;%CLASSPATH% *.java

   Note The complete pathname for **javac.exe** is not necessary if your system PATH variable already includes the corresponding **javac.exe**.

   Caution You must replace all directory name (package) occurrences; if you receive errors messages stating that certain symbols could not be resolved or found, check the files listed in the error statements to ensure that all package names are correct.
7. Create a new JAR file containing your customized Step Processor.
8. Sign the JAR file.
9. Deploy the updated Step Processor applet on the Web WorkFlo server.
10. As a test, run the new applet to make sure you can use it.

If the new custom Step Processor works, you are ready to customize the Step Processor application/applet code (see below).

Customize the Code

This section describes how to add user interface controls to the custom Step Processor by adding simple fields to indicate when the Step Processor was sent (launch information) and who sent it (originator information).

1. Using a text editor, open the renamed <prefix>stepPanel.java file (where the <prefix> is the prefix you appended to the file earlier).
2. Define the new controls by adding the following statements at the end of the DECLARE CONTROLS sections (near the bottom of the file):

```
//Create labels for the launch date and originator information.

javax.swing.JLabel sentLabel = new javax.swing.JLabel();
```

```
javax.swing.JLabel fromLabel = new javax.swing.JLabel();

//Create label objects for the launch date and originator information.

filenet.vw.toolkit.runtime.step.beans.VWLabel vwSentLabel = new
filenet.vw.toolkit.runtime.step.beans.VWLabel();
filenet.vw.toolkit.runtime.step.beans.VWLabel vwFromLabel = new
filenet.vw.toolkit.runtime.step.beans.VWLabel();
```

3. Locate the following statement, and change the position values to move the icon out of the current position to make room for the new fields.

```
iconLabel.setBounds(516,48,216,108);
```

   Modify the values to change the location of the icon. For example you might enter the following values:

```
iconLabel.setBounds(20,48,216,108);
```

4. Define the launch date information by adding the following statements to the code, in the INIT_CONTROLS section:

```
//Display the label text on the JPanel to specify what the launch date information means.

sentLabel.setText("Launched on:");
add(sentLabel);
sentLabel.setBounds(320,84,100,24);

//Retrieve and display the launch date for Step Processor.

vwSentLabel.setParameterName("F_LaunchDate");
add(vwSentLabel);
vwSentLabel.setBounds(430,84,100,24);
```

5. Define the originator information by adding the following statements to the code:

```
//Display the label text on the JPanel to indicate what the originator information means.

sentLabel.setText("Step Originator:");
add(fromLabel);
fromLabel.setBounds(320,120,100,24);

//Retrieve and display the originator information.

vwFromLabel.setParameterName("F_Originator");
add(vwFromLabel);
vwFromLabel.setBounds(430,120,180,24);
```

6. Save the changes and close the file.
7. Compile the samples by entering a command similar to the following (modify for the appropriate JDK):

```
C:\jdk1.3.1\bin\javac.exe -classpath .;C:\<your directory>\pw.jar;%CLASSPATH% newStepApplet.java
```

Alternatively, enter a command similar to the following to compile all of the classes in the directory at once:

C:\jdk1.3.1\bin\javac.exe -classpath .;C:\<*your directory*>\pw.jar;%CLASSPATH% *.java

Caution You must replace all directory name (package) occurrences; if you receive errors messages stating that certain symbols couldn't be resolved or found, check the files listed in the error statements to ensure that all package names are correct.

8. Create a new JAR file containing your newly customized Step Processor.
9. Replace the JAR file you installed earlier on the Web WorkFlo server, with this updated version.
10. As a test, run the new applet to make sure the changes are visible and that the applet works.

Java Step Processor Sample Files

The Java Step Processor sample files are located in the **\Developer Files\Samples\vwpanel\samplestep** directory. Refer to Install Panagon eProcess Toolkit for information on installing the appropriate development files on your system.

The directory contains the following files:

File name (and location)	Description
VWSampleStepApplet.java	For applets hosted in .ASP files. Implements methods for loading the window by using the VWResource and VWImageLoader classes. Creates and initializes the processor application. Loads the window, sets the title, establishes a VWSession, and calls the main user interface panel.
VWSampleStepApplication.java	For applications. Implements methods for loading the window by using the VWResource and VWImageLoader classes. Creates and initializes the processor application. Loads the window, sets the title, establishes a VWSession, and calls the main user interface panel.
VWSampleStepPanel.java	Implements the user interface visual elements for the Step Processor. Initializes the controls for the panel.
\resources\VWResource.java	Acts as a resource provider for the processor. The default resource is the title string for the applet.
\images\VWImageLoader.java	Loads images for the VWSampleStepPanel container.
\images\icon.gif	Image file for VWSampleStepPanel container.
\images\step.gif	Image file for the VWSampleStepPanel container to indicate that this is a Step Processor.

Working with the VWAttachmentPanel

The following code demonstrates one way of using the VWAttachmentPanel bean to access attachments, stored in Content Services or Image Services libraries, from Java-based Step Processors or Launch Step Processors.

This method of instantiating the VWAttachmentPanel is useful for developers who are not using an IDE (such as Visual Cafe, JBuilder, Forte, etc.) capable of supporting the RAD drag-and-drop components packaged in the **vwpanel.jar** file, or for anyone interested in understanding how to retrieve attachments apart from the VWTabbedPane control. If you are using the drag-and-drop components in an IDE, the code for instantiating the VWAttachmentPanel component is added automatically when you drag the component onto a form.

The VWTabbedPane component includes the VWAttachmentPanel, and the component supplies other menu items automatically. This code demonstrates how to use the VWattachmentPanel control separately from the VWTabbedPane component.

1. To initialize the bean properly pass the following information:

- A VWStepElement object
- A Frame object (the containing parent frame)
- A valid VWSession object

2. The following sample code assumes that you declare and initialize these three variables near the top of the class definition.

```
VWStepElement    vwStepElement = null;
Frame            parentFrame = null;
VWSession        vwSession = null;
```

3. Next, instantiate the VWAttachmentPanel bean after testing for the VWStepElement field type:

```
JComponent       newJComponent = null;

if (vwStepElement.getParameters(VWFieldType.FIELD_TYPE_ATTACHMENT,
VWStepElement.FIELD_USER_DEFINED) != null)
{
  newJComponent = (JComponent)Beans.instantiate(getClass().getClassLoader(),
  "filenet.vw.toolkit.runtime.step.beans.VWAttachmentPanel");

  if (newJComponent != null)
    {

\\While it is not shown here, you should include code to add the JComponent to the
\\containing VWPanel to expose the attachment list to the end user.

     if (newJComponent instanceof IVWStepProcessorComponent)
     ((IVWStepProcessorComponent)newJComponent).init(vwStepElement, parentFrame, vwSession);
    }
```

```
}
```

Developing Custom Java Processors - Applications vs. Applets

Before you deploy a custom Java Step or Launch Processor, you will need to decide whether to deploy it as an application or an applet (there may be some advantages to deploying it as an application). For the purposes of this discussion, these terms may be defined as:

- **Applet** — an applet is a program that is launched and run within the context of a web browser. (In a Java Step Processor Toolkit context, applets refer to classes that extend JApplet, or FileNET's VWBaseAppLauncherApplet class, which is itself an extension of JApplet.)
- **Application** — an application is a Java application invoked by its package/class name. (In a Java Step Processor Toolkit context, applications refer to classes that extend JFrame, or FileNET's VWBaseLaunchableApplication class, which is itself an extension of JFrame.)

This topic explores this issue in terms of:

Applications vs. Applets — General Guidelines

The following general guidelines apply when considering whether to deploy your custom Java Step or Launch Processor as an application or an applet:

- In general, it is optional as to whether you deploy a Java Step or Launch Processor as an application or an applet. The sample or out-of-the-box Step and Launch Processors included with the Java Toolkit will each run as either an application or an applet.
- If a Java Step Processor is to be opened from an email notification message, you must implement an applet class. In other words, if you deploy as an application, the Step Processor must implement an applet class in order to open the step assignment from an email notification message. On the other hand, if a Step Processor is to be opened from an email notification message and you deploy it as an applet, there is no requirement to implement an application class. Since Launch Processors cannot be opened from an email notification message, there is no need to implement an applet class if you are deploying a Launch Processor as an application.

Why Deploy Your Step Processor as an Application

There are a number of advantages in deploying a custom Java Step or Launch Processor as an application. These include:

- Deploying the Step Processor as an application provides a better connection between the Step Processor and the Personal Work Manager (PWM). When you complete or close a step when the Step Processor is deployed as an application, the application notifies the PWM that an event has occurred, causing the PWM to automatically refresh its display. When you complete or close a

step when the Step Processor is deployed as an applet, the user must manually refresh the PWM in order to refresh its display.

- Deploying the Step Processor as an application enables a session to be shared; that is, the user is not required to login again. This is true if your application class implements the IVWLaunchableApp interface, either directly or by extending the VWBaseLaunchableApplication class as in the Java Step and Launch Processor sample code.
- Deploying the Step Processor as an application forces popup dialogs to be modal. This prevents users from doing something else outside the dialog box while the dialog entry is unresolved. Deploying the Step Processor as an applet does not provide dialog modality.
- Deploying the Step Processor as an application allows the application to use the same instance of the Java plug-in. Deploying the Step Processor as an applet requires the applet to create a new instance of the Java plug-in, thereby slowing performance.

Modify the HTML Step Processor

This topic describes how to customize the default HTML Step Processor to meet unique workflow requirements.

While no Integrated Development Environment (IDE) requirement for ASP development exists, these procedures assume you are using Visual InterDev and that you have a solution already set up similar to the one described in the Developing for ASP section of this guide. An alternative to using Visual InterDev is working directly in the <drive>...\Program Files\FileNET\IDM\Web\IDMWS directory structure on your Web WorkFlo server. If you know ASP development requirements and you want to make minor changes to specific files, working in the directory structure might be the best solution. The primary disadvantage is you will be working in an active web server directory structure, so mistakes are hard to correct.

Caution The HTML processors are designed to reuse some shared files, which are located in the \WF_Html_Toolkit directory. Do not modify the files in the \WF_Html_Toolkit directory. Modifying the shared files might cause problems with Step Processor and Launch Step Processor shipped with Web WorkFlo.

Customize the default HTML Step Processor by performing the following steps:

1. Create a new directory under the \Redist\WF_Step directory (at the same level as the \HTML directory). If you are using Visual InterDev, you can add a new Folder.
2. Name the directory according to the function of the processor. (This procedure will use the directory name \new_HTML_Step.)
3. Copy the contents of the \Redist\WF_Step\html directory into the \new_HTML_Step directory.
4. Rename the step_main.asp file to newstep_main.asp. If you are not using Visual InterDev, you must change the file attribute from Read-only to Read/Write.
5. Open the newstep_main.asp file.
6. Modify the .ASP files as needed. (Refer to the HTML Step Processor Relationships topic for more information on Step Processor file dependencies and the frameset implemented in the file.)
7. If you need to modify any of the files in the \WF_Html_Toolkit directory, copy the needed files into the directory you created for your new processor.
8. Change all references to the file(s) in the modified processors.
9. Add the Step Processor to the workflow. (Refer to *Configure Step Processors* in the *Management* section of the Configuration Console help for more information on adding Step Processors to a workflow.)
10. Test your new Step Processor pages from your application development server.

User interface Modifications

The default **step_main.asp** file includes the colorOpt variable for controlling the default color of the Step Processor tabs. The default colorOpt value is a hexadecimal value of "#657FD2" (corresponding to a RGB color model). The variable is passed to the stepInfo.asp, tab.asp, and tabContent.asp files.

Modify the HTML Launch Step Processor

This topic describes how to customize the default HTML Launch Step Processor to meet unique workflow requirements.

While no Integrated Development Environment (IDE) requirement for ASP development exists, these procedures assume you are using Visual InterDev and that you have a solution already set up similar to the one described in the Developing for ASP section of this guide. An alternative to using Visual InterDev is working directly in the <drive>...\Program Files\FileNET\IDM\Web\IDMWS directory structure on your Web WorkFlo server. If you know ASP development requirements and you want to make minor changes to specific files, working in the directory structure might be the best solution. The primary disadvantage is you will be working in an active web server directory structure, so mistakes are hard to correct.

Caution The HTML processors are designed to reuse some shared files, which are located in the \WF_Html_Toolkit directory. Do not modify the files in the \WF_Html_Toolkit directory. Modifying the shared files might cause problems with Step Processor and Launch Step Processor shipped with Web WorkFlo.

Customize the default HTML Launch Step Processor by performing the following steps:

1. Create a new directory under the \Redist\WF_Launcher directory (at the same level as the \HTML directory). If you are using Visual InterDev, you can add a new Folder.
2. Name the directory according to the function of the processor. (This procedure will use the directory name \new_HTML_Launcher.)
3. Copy the contents of the \Redist\WF_Launcher\html directory - including the images directory - into the \new_HTML_Launcher directory.
4. Rename the launcher_main.asp file to newlauncher_main.asp. If you are not using Visual InterDev, you must change the file attribute from Read-only to Read/Write.
5. Open the newlauncher_main.asp file.
6. Modify the .ASP files as needed. (Refer to the HTML Launch Step Processor Relationships topic for more information on launch processor file dependencies and the frameset implemented in the file.)
7. If you need to modify any of the files in the \WF_Html_Toolkit directory, copy the needed files into the directory you created for your new processor.
8. Change all references to the file(s) in the modified processors.
9. Add the Launch Step Processor to the workflow. (Refer to *Configure Step Processors* in the *Management* section of the Configuration Console help for more information on adding Launch Step Processors to a workflow.)
10. Test your new Step Processor pages from your application development server.

Note You might want remove image files you know you will not be using, copy only those image file you are sure to use, or add image files to the \Images sub-directory in your new directory structure.

User Interface Modifications

The default **launcher_main.asp** file includes the colorOpt variable for controlling the default color of the Launch Step Processor tabs. The default colorOpt value is a hexadecimal value of "#669966" (corresponding to a RGB color model). The variable is passed to the tab.asp, tabContent.asp, and stepInfo.asp files.

Modify User Interface Files

This topic describes how to modify, in general terms, the following user interface files used by the default HTML Step Processors and Launch Step Processors. In all liklihood, you will not need to modify most of the user interface files; in the following sections, an asterisk (*) following the file name indicates a file you are most likely to modify.

Caution You can add new functions or extend the existing function in these files; however, you must not delete or alter the existing functions

Many of the files listed in this topic directly referenced in the step_main.asp or launcher_main.asp files (or the modified versions of those files). Refer to both the HTML Step Processor File Relationships and the HTML Launch Step Processor File Relationships topics for more information on the file dependencies.

arrayEdit.asp

Called by: dataField.asp file (only for data fields that are arrays)

The arrayEdit.asp file provides Javascript functions for loading an array, displaying the array name, checking the field type (integer, string, boolean, float, and time), adding and inserting array elements, among other functions. The embedded form uses a table to display the field information and post the information to the dataField.asp file.

---

attachment.asp

Called by: Attachments tab (which is implemented in the tabContent.asp file)

The attachment.asp file acts as a parent container for the attToolbar.asp and attachmentTabke.asp files. The file implements the following frameset:

```
<frameset rows="35,*" border="0" framespacing="0" frameborder="NO">
  <frame name="AttToolbar" src="Redist/WF_Html_Toolkit/UI/attToolbar.asp" scrolling="no" noresize >
  <frame name="AttTable" src="Redist/WF_Html_Toolkit/UI/attachmentTable.asp?colorOpt=<%=TabColor%>"
frameborder=1 scrolling="auto">
</frameset>
```

The tab display variable colorOpt passes the value of the VBScript variable <%TabColor%>. If no color is specified, the default highlight color is blue. The color of the highlighted tab can be selected by the following query string statement:

TabColor = Request.QueryString("tabColor")

The tabColor variable value is used to set distinguishing colors for both the HTML Step and Launch Step Processors.

---

attachmentTable.asp*

Called by: the frameset implemented in the attachment.asp file

The attachmentTable.asp file implements Javascript functions for dynamically determining attachment properties and constructing a HTML-formatted table, using Javascript, to display the properties.

The functions test for the attachment type and construct a table which includes and displays the attachment name, attachment item description, the library where the attachment resides, and the attachment document ID, which is the Images Services or the Content Services id number.

You can modify the dynamically constructed <A HREF> tags and extend the function implemented in the file to change link behavior, or you can add additional attachment information and extend the number of columns or rows constructed for each attachment.

---

attToolbar.asp

Called by: the frameset implemented in the attachment.asp file

The attToolbar.asp includes the ToolbarButton.asp file and implements functions that allow a user to select an attachments (like documents, folders, or stored searches) in the attachmentTable.asp file to view, checkout, cancel checkouts, and view version information. The toolbar images and functionality changes depending on the objected selected in the attachmentTable.asp file.

Most of the functions for opening attachments are handled by the .asp modules in the \WF_FileOpen directory. Specifically, the attachments functionality is contained in the FileOpen.js/fileopen.asp and Otherselect.js/OtherSelect.asp files, which are included in this file at runtime. You must know the IDM Web Services Javascript objects thoroughly to extend these modules.

You can extend the existing functions or add additional event checking.

---

dataField.asp*

Called by: Data Fields tab (which is implemented in the tabContent.asp file)

The dataField.asp file constructs a table row by row, using Javascript, to display step element parameter information. You can change the table creation, or you can extend the function for editing the array.

For parameters that are arrays, the HTML includes an image-based link in the table. When an end user clicks the image, the link invokes the arrayEdit.asp file, as a new window instance.

---

milestone.asp

Called by: tab.asp, milestone.asp, and open_milestone.js

The milestone.asp file constructs a table to display the milestone information, like the milestone name, the associated message, and the time the milestone was reached.

---

generalInfo.asp

Called by: tab.asp and tabContent.asp

The generalInfo.asp file provides Javascript functions for selecting and displaying the tab-related information used in the HTML Step Processor and Launch Step Processor.

---

reassign.asp*

Called by: toolbar.asp (if the step can be reassigned)

The reassign.asp file provides a Javascript pull-down list for reassigning a workflow step to a different participant. The file implements functions for reading participant arrays, sorting the elements alphabetically, and creating an <Option> tag (for the <Select> pull-down list) for each array element. You can modify the way the participant information is selected or displayed.

---

resource.asp*

Called by: stepElementInit.asp, toolbarHelper.asp, arrayEdit.asp, attachment.asp, attachmentTable.asp, attToolbar.asp, dataField.asp, generalInfo.asp, milestone.asp, reassign.asp, stepInfo.asp, tab.asp, workGroup.asp, workGroupEdit.asp

The resource.asp file provides variables associated with the strings of displayed text for the user interface components listed above; the varibles are assigned string values in this file. Alter the strings to localize the text.

---

stepInfo.asp

Called by: step_main.asp and launcher_main.asp

The stepInfo.asp page constructs a table for displaying instructions. The following query string statement controls the background color of the instruction label:

ColorOpt=Request.QueryString("colorOpt")

The colorOpt variable, which is passed from the main_step.asp file, the specified value is "#657FD2". If no value exists, the color defaults to "Blue".

---

tab.asp*

Called by: step_main.asp and launcher_main.asp

The tab.asp file works with tabContent.asp to implement the default tabs: **General** tab, Attachment tab, Data Fields tab, Workflow Groups tab, and **Milestones** tab. The file selects the tabs displayed in the tabContent.asp file.

The tabs are evaluated in a bit-wise (binary) fashion; each default tab is assigned the following bit value:

- General tab = 0 (0000)
- Attachment tab = 1 (0001)
- Data Field tab = 2 (0010)
- Work Groups tab = 4 (0100)
- Milestone tab = 8 (1000)

You can modify how the tabs display by using bit masking. For example, a value of 1 (0001) displays the **General** and **Attachment** tabs, and a value of 2 (0010) displays the **General** and **Data Fields** tabs. However, a value of 6 (0110) displays the **General**, **Data Fields** and **Work Group** tabs. A value of 15 (1111) displays all tabs. Regardless of the tab being displayed, all of the tabs display left aligned.

If you want to add a tab, perform the following general steps:

1. Modify the tab.asp file, or copy the file to a local directory - depending on whether or not you want the new tab to

be available to all processors or only the one you are modifying.

2. Associate a bit value to represent the new tab. For example, a fifth tab must have a value of 16 (10000), and a sixth tab must have a value or 32 (100000).
3. Modify the local tabContent.asp to handle the new tab. The new tab must be able to reference all other tabs.

The current default function is similar to the following:

```
function onTabClick(sel)
{
       if (parent.TabContent.updateStep() == false)
            return;

        var htmlString = null;
  if (sel == 0)
   {
              htmlString = "generalInfo.asp?tabColor=<%=colorOpt%>";
  }
       else if (sel == 1)
      {
              htmlString = "attachment.asp?tabColor=<%=colorOpt%>";
   }
       else if (sel == 2)
      {
              htmlString = "dataField.asp?tabColor=<%=colorOpt%>";
    }
       else if (sel == 3)
      {
              htmlString = "workGroup.asp?tabColor=<%=colorOpt%>";
    }
       else if (sel == 4)
      {
              htmlString = "milestone.asp?tabColor=<%=colorOpt%>";
    }
       if (htmlString != null)
 {
              parent.TabContent.location = htmlString;
        }
       parent.TabHeader.location = "tab.asp?colorOpt=<%=colorOpt%>&tabSel=" + sel;
}
```

---

tabContent.asp*

Called by: step_main.asp and launcher_main.asp

With the tab.asp file, the tabContent.asp file implements functionality for bitwise comparison of the tabs for the step. The General tab always display, the remaining tabs are selected depending on the existence the required elements, such as attachments, or workgroups, or milestones.

If you want to add a tab, you must modify this file along with the tab.asp file.

---

toolbar.asp

Called by: step_main.asp - or modified versions of the file. The launcher_main.asp file uses a different version of the file, which is located in the \WF_Launcher\HTML directory.

Modifying this file will allow you to rearrange, replace, remove, and add buttons. All button event handlers are implemented and defined in the toolbarHelper.asp file.

- If you are modifying a toolbar that is available to all step or launch processors, you must also add a button and an associated handler function to the toolbarHelper.asp file.
- If you are modifying a toolbar for only one processor, copy the toolbar.asp file to the directory containing the customized step or Launch Step Processor. Add the button and button event handler in the local copy of the toolbar.asp. Modify your Step Processor page(s) to reference to the local toolbar.asp.

---

workGroup.asp*

Called by: Workflow Groups tab (which is implemented in the tabContent.asp file)

The workGroup.asp file creates a table automatically, using Javascript, to display the work group name and participant information for each workgroup defined for the workflow.

Each dynamically-constructed row includes an includes an image-based link that allows the Javascript onClick event to invoke the workGroupEdit.asp file. When an end user clicks the image, the link invokes the workGroupEdit.asp file, as a new window instance.

You can modify how the work group information displays by changing the HTML-formatted tables.

---

workGroupEdit.asp

Called by: workGroup.asp

The workGroupEdit.asp file implements functions for determining and dynamically creating a HTML-formatted table to edit the work group participants for the work groups.

The file implements functions for reading work group participant arrays, sorting the elements alphabetically, and creating a check box interface for selecting specific array elements.

Modify Core Files

This topic describes how to modify, in general terms, the following core shared files used by the default HTML Step Processors and Launch Step Processors. In all liklihood, you will not need to modify most of these core files; in the following sections, an asterisk (*) following the file name indicates a file you are most likely to modify. This topic does not include the server-side include file, which are located in the same directory at these files.

Caution You can add new functions or extend the existing function in these files; however, you must not delete or alter the existing functions.

launchStepElementInit.asp

Similar to the stepElementInit.asp file. The file includes the following Web WorkFlo helper files:

- loadFields.inc
- loadWorkGroup.inc
- loadAtt.inc

The file uses the JiGlue COM Bridge to work with VWAttachment objects in WorkFlo Services. Refer to this file to determine how to create and access attachments that can be viewed in workflow processing.

Like the stepElementInit.asp file, the launchStepElementInit.asp implements server-side VBScript functions and client-side Javascript functions for determining and initializing system field values for the user interface components.

Additionally, the file uses the Web Services client-side, Javascript, IDMWSC_Result object to implement a Javascript switch structure. Refer to Locate the Panagon Web Services Toolkit Help for more information on using the Panagon Web Services components.

---

launchStepProcessor.asp

Includes the following Web WorkFlo helper files:

- stepProcessorAttHelper.asp
- stepProcessorFieldHelper.asp
- stepProcessorWGHelper.asp

Refer to the file to examine the remaining referenced and included files.

The launchStepProcessor.asp file is a top-level page. The file implements VBScript functions for parsing XML step element strings composed by on the client system. Additionally, the file includes a template for the XML schema being used to parse the launch step element string and includes a description of the assumptions. For reference purposes, the well-formed XML schema is listed here:

```
<step>
```

```
<comment>...</comment>
<subject>...</subject>
<wfId>...</wfId>
<selResponse>...</selResponse> -- Optional reponses

<att>
    <name>...</name>
    <isArray>0</isArray>  -- 0 = non-array, or 1 for array.
    <idmAtt>
       <name>...</name>
       <desc>...</desc>
       <type>...</type>
       <lib>...</lib>
       <libType>...</libType>
       <id>...</id>
    </idmAtt>
... -- one or more IDM attachment
</att>
<att>
...  -- One or more attachment objects
</att>

<field>
    <name>...</name>
    <desc>...</desc>
    <type>...</type>
    <value>...</value>
    <isArray>...</isArray>
</field>
<field>
...   -- One or more field
</field>

<workgroup>
     <name>...</name>
     <participants>...</participants>
</workgroup>
<workgroup>
... -- One or more work groups
</workgroup>

</step>
```

Several other files (like stepProcessorAttHelper.asp, stepProcessorFieldHelper.asp, and stepProcessorWGHelper.asp) help to parse the XML string.

stepElementInit.asp

Includes the following Web WorkFlo helper files:

- loadFields.inc
- loadWorkGroup.inc
- loadAtt.inc
- loadMilestones.inc

Refer to the file to examine the remaining referenced and included files.

The stepElementInit.asp page implements VBScript server-side functions for retrieving step element properties used to populate the client-side components.

Additionally, the file uses the Web Services client-side, Javascript, IDMWSC_Result object to implement a Javascript switch structure. Refer to Locate the Panagon Web Services Toolkit Help for more information on using the Panagon Web Services components.

---

stepProcessor.asp

Includes the following Web WorkFlo helper files:

- stepProcessorAttHelper.asp
- stepProcessorFieldHelper.asp
- stepProcessorWGHelper.asp

Refer to the file to examine the remaining referenced and included files.

The stepProcessor.asp file is a top-level page. The file implements VBScript helper functions for retrieving and restructuring XML step element strings. The implemented functions assume that an VWSession object already exists.

The file includes a template for the XML schema being used to parse the step element information. The full XML schema is shown in the launchStepProcessor.asp file description (above).

---

stepProcessorAttHelper.asp

The stepProcessorAttHelper.asp file implements server-side VBScript functions for parsing the XML string generated on the client system and saving the property information contained in the Content Services or Image Services attachment (<idmAtt>...</idmAtt>) tag set of the well-formed XML string.

The file uses the JiGlue COM Bridge to work with VWAttachment objects in WorkFlo Services.

```
<att>
```

```
    <name>...</name>
    <isArray>0</isArray>  -- 0 = non-array, or 1 for array.
    <idmAtt>
       <name>...</name>
       <desc>...</desc>
       <type>...</type>
       <lib>...</lib>
       <libType>...</libType>
       <id>...</id>
    </idmAtt>
</att>
```

Note The full XML schema is shown in the launchStepProcessor.asp file description (above).

---

stepProcessorFieldHelper.asp

The stepProcessorFieldHelper.asp file implements server-side VBScript functions for parsing the XML string generated on the client system and changing the system field property information contained in the (<field></field>) tag set of the well-formed XML string.

```
<field>
    <name>...</name>
    <desc>...</desc>
    <type>...</type>
    <value>...</value>
    <isArray>...</isArray>
</field>
```

Note The full XML schema is shown in the launchStepProcessor.asp file description (above).

---

stepProcessorWGHelper.asp

The stepProcessorWGHelper.asp file implements server-side VBScript functions for parsing the XML string generated on the client system and changing property information contained in the work group (<workgroup></workgroup> tag set of the well-formed XML string.

```
<workgroup>
     <name>...</name>
     <participants>...</participants>
</workgroup>
```

Note The full XML schema is shown in the launchStepProcessor.asp file description (above).

---

toolbarHelper.asp

The toolbarHelper.asp, working with the step.js file, defines the toolbar button behaviors. The toolbarHelper.asp file calls the toXML() method defined in the step.js file to generate an XML string, which is sent from the client to the server.

Modify Utility Files

Do not modify the shared utility unless you have a compelling reason to do so. This topic describes, in general terms, why you might modify the utility files.

Caution You can add new functions or extend the existing function in these files; however, you must not delete or alter the existing functions.

utility.js

Modify and use this file for client-side sorting. If your application requires each user to frequently sort field arrays, you should reference and extend this file. All of the sorting functionality is handled by the locally-cached version of this file. When relying on the client-side functionality implemented in this file, only one user can access the sorting.

---

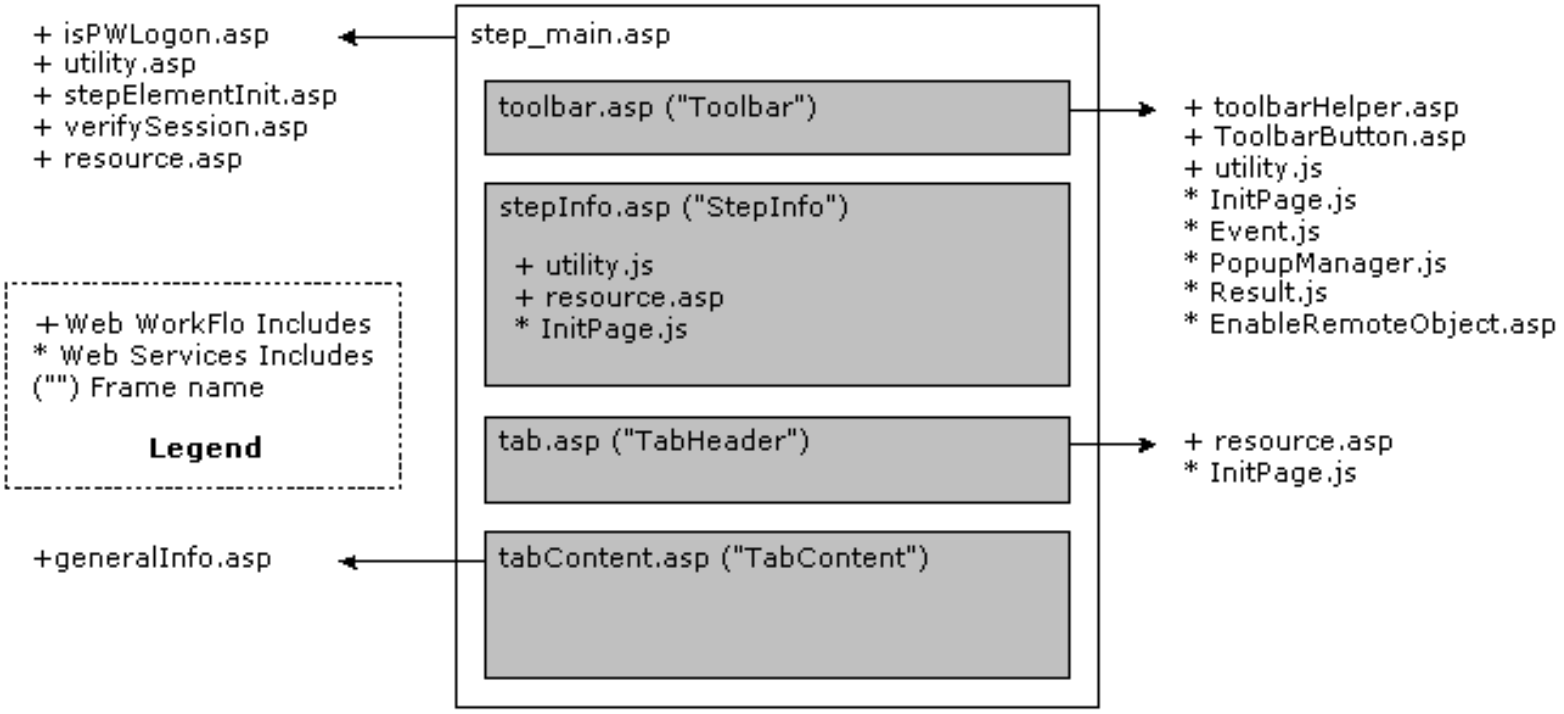
utility.asp

Modify and use this file for server-side sorting. If your application requires that a single sorting operation is distributed across the enterprise - among many users, then reference and extend this file. All sorting functionality is handled by the file on the server, so using this file for frequent sorting will cause the server operation to degrade as the load increases.

HTML Step Processor File Relationships

This topic illustrates the relationships and dependencies of the HTML Step Processor. The figure shows both the Panagon Web Services and Web WorkFlo files included in, or referenced by, the default **step_main.asp** file.

Note The figure does not show referenced images.



For descriptions of the Web Services files, see the Referenced Panagon Web Services Files and the Locate the Panagon Web Services Toolkit Help topics. For descriptions of the Web WorkFlo files, see the Referenced Panagon Web WorkFlo Files and the HTML processor toolkit file topics.

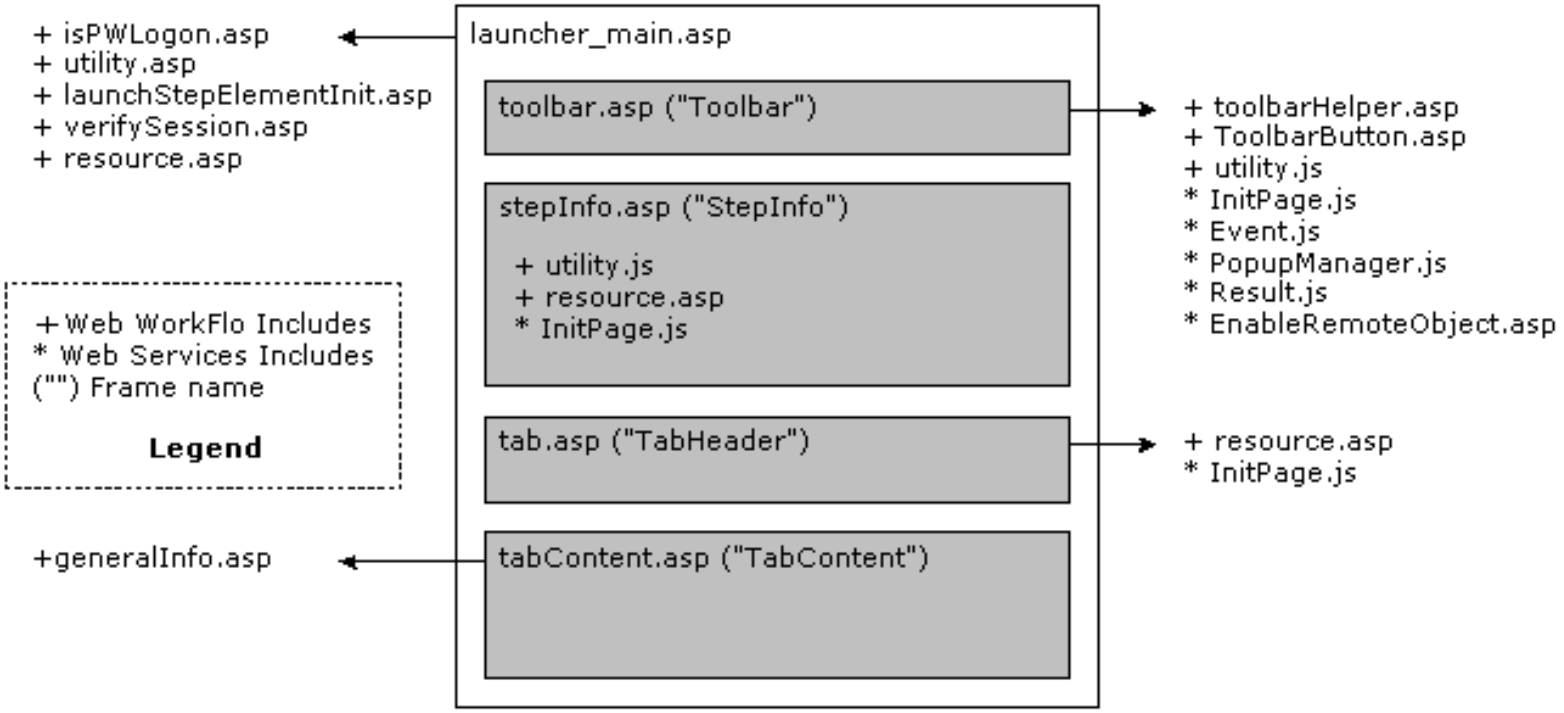
The following table lists the file names, locations, and frame names for all of files directly referenced by the **step_main.asp** file.

File name (and location)	Frame
WF_Html_Toolkit\UI\toolbar.asp	Toolbar
WF_Html_Toolkit\UI\stepInfo.asp	StepInfo
WF_Html_Toolkit\UI\tab.asp	TabHeader
WF_Html_Toolkit\UI\tabContent.asp	TabContent

HTML Launch Step Processor File Relationships

This topic illustrates the relationships and dependencies of the HTML Launch Step Processor. The figure shows both the Panagon Web Services and Web WorkFlo files included in, or referenced by, the default launcher_main.asp file.

Note The figure does not show referenced images.



For descriptions of the Web Services files, see the Referenced Panagon Web Services Files and the Locate the Panagon Web Services Toolkit Help topics. For descriptions of the Web WorkFlo files, see the Referenced Panagon Web WorkFlo Files and the HTML processor toolkit file topics.

The table (below) lists the file names, locations, and frame name for all of files directly referenced by the **launcher_main.asp** file. The Toolbar and StepInfo frames use specialized .ASP files. The toolbar is different from that used in the HTML Step Processor. The StepInfo frame displays only the launch subject. The **stepInfo.asp** file implements functions for initializing the workflow subject and comments.

File name (and location)	Frame
WF_Launcher\html\toolbar.asp	Toolbar
WF_Html_Toolkit\UI\stepInfo.asp	StepInfo
WF_Html_Toolkit\UI\tab.asp	TabHeader
WF_Html_Toolkit\UI\tabContent.asp	TabContent

Add Custom Processors to the Workflow

Once you have either created a new Step Processor or Launch Step Processor or extended the existing versions, you must add the custom Processor to the workflow before an end user can access them.

Caution If you upgrade to a new Panagon Web Services version, you **must** redeploy all existing custom Step and Launch Processors (for additional information, see the Deploying a Custom Java Step or Launch Processor procedure).

If you add the Step Processor or Launch Step Processors while Designer is running, the new processor is not available until the next time you start Designer. Stop Designer before adding custom processors. (Refer to *Configure step processors* in the *Management* tab of the Configuration Console help for more information on adding Step Processors or Launch Step Processors to a workflow.)

The steps for adding a custom processor differ very little between HTML (ASP) and Java processors. The primary difference is that for applets hosted in ASP and HTML Step Processors and Launch Step Processors, you need not specify the package in the *Class Name* field.

Note Perform the following steps on a client system that can connect to the Web WorkFlo server on which you installed the modified .JAR file.

Add a custom processor by performing the following general steps:

1. Connect to your eProcess Server, and open the Configuration Console.
2. Log on to the proper region. Locate and select the router associated with the region.
3. Highlight the router name, and right-click the highlighted name. Select **Properties**.
4. From the **Isolated Region Properties** dialog box, select the **Step Processor Info** tab.
5. Click add and enter the following step definition information:
    - *Type*: Enter either **Step** or **Launch**.
    - *Name*: Enter the name you want to display in the Designer.
    - *Language*: Enter **HTML** for all Step Processors that are HTML, ASP, and applets hosted in ASP. Enter **Java** only when specifying a Java application.
    - *Location*: Enter the location and name of the file for the modified **step_main.asp** or **launcher_main.asp** files.
    - *Width*: Enter the width in pixels.
    - *Height*: Enter the height in pixels.
    - *Class Name*: Enter the class name only if you specified **Java** earlier. The class name is the package name containing the modified .JAR file. For example, if the .JAR was created in the **\newStep** directory and the class is called `newStepApplet`, the package name is `newstep.newStepApplet`.
6. Click **OK**, and commit the changes.
7. Close Configuration Console.
8. Open Designer. The newly added Step Processor or Launch Step Processor should display as one

of the available **Step Processor** options.

Signing a JAR File

Web WorkFlo and WorkFlo Services require that you digitally sign custom step processor Java ARchive (JAR) with either a Netscape Object Signing Certificate or a Sun Java Object Signing Certificate.

Developers who create custom step processors can obtain a digital certificate to sign their JAR files from either VeriSign Inc. (http://digitalid.verisign.com/developer/nos_pick.htm) or Thawte Consulting (http://www.thawte.com/). (The default JAR files are signed with a Netscape Object Signing Certificate.) Custom signing has been tested in custom step processor JAR files with this type of certificate. A Netscape Object Signing Certificate is the recommended option. With a single certificate, developers can create and sign as many custom step processor JAR files as needed.

Signing a Java ARchive (JAR) File

The following procedure details how to create a JAR file signed with the Netscape Object Signing Certificate. The procedure describes how to use the Netscape signing tool (signtool) in a general fashion. For more information, visit the Netscape Developer's Site (http://developer.netscape.com/software/signedobj/jarpack.html).

Note If you purchase a Sun Java Object Signing Certificate, you must use the Sun Microsystems JAR signing tool: jarsign. You must use a different procedure from that listed below. (Sun offers an on-line tutorial demonstrating how to use the jarsign tool at http://java.sun.com/docs/books/tutorial/jar/sign/index.html.)

1. Create a new directory on the your development system. (For example, C:\certificate.)
2. Copy the files supplied by VeriSign or Thawte to the new directory.
3. Create a second new directory. (For example, C:\Sign.)
4. Copy the contents of the files to include in the JAR file to the second directory. (For example, C:\Sign.) Make sure you include the MANIFEST.MF file in a META-INF subdirectory of the second newly created directory. (For example, you would place the MANIFEST.MF file in the C:\Sign\META-INF directory.)
5. Open a command Prompt, and change to the directory directly above your C:\sign directory. (For example, C:\.)
6. Enter a command similar to the following:

C:\certificate\signtool -d"C:\certificate" -k"<Your_company_name>" -Z"<Your_jar_file_name>" -p"<Your_password>" sign

   where <Your_company_name> is replaced by the company name as stated by your certificate provider, <Your_jar_file_name> is replaced by the name you wish to give the JAR file, <Your_password> is replaced by your password supplied by your certificate provider, and sign is the name of the directory containing the files your wish to have in the signed JAR file.

7. Once signtool finishes, verify that the JAR was signed properly by entering the following command:

```
C:\certificate\signtool -d"C:\certificate" -w Your_jar_file_name
```

where <Your_jar_file_name> is replaced by the name you wish to give the JAR file.

Modify Email Notification Files

The eProcess Services email notification (.MSG) files are text files containing HTML-formatted content. Since the notification files contain HTML-formatted content, you can modify the look of the resulting output to match your corporate identity or to meet specific workflow requirements; however, you cannot add notification files, and you cannot rename the existing files. In addition to the U.S. English versions, localized versions of the messages files exist for French, German, Italian, and Spanish.

Topics for modifying email notification files include:

- Modifying eMail Notification Files Procedure
- eMail Notification Types (including filenames and descriptions)
- Optional eMail Notification Variables
- Using Secure Sockets Layer (SSL)

Modifying eMail Notification Files Procedure

Using eProcess Services email notification requires that you first enable logging for the *Exception* and *Begin Operation* categories. Configure the logging options in the Configuration Console. (Refer to the *Configure event log options* topic, in the *Event and Statistics* tab of the in the Configuration Console help, for more information.)

Before modifying the email notification files, you should expose the associated system fields, if they have not already been exposed or are not exposed by default. Exposed fields are system and data fields in a roster, queue, or event log that have been made available for searches and sorting.

The email notification files contain placeholder variables embedded in the HTML. The variables values are supplied by WorkFlo Services at runtime, where a given field variable obtains its values from the field that is named by the string following the "$" prefix in the field variable name. The WorkFlo Services Notification process caches the notification files when the process starts. When invoked, the WorkFlo Services Notification process retrieves the necessary data from the notification information database, populates the variables with the relevant field information, and dispatches the file via the SMTP service that is specified in the **vwserver.ini** file. For information on optional email notification variables, see Optional eMail Notification Variables.

Modify the email notification files by performing the following general procedure:

1. On the WorkFlo Services server, navigate to the **\fnsw_loc\sd\msg\2** directory (for ActiveX). (On UNIX servers, navigate to the **/fnsw/loc/sd/msg/2** directory.) Note that for Open Client, go to the **\fnsw_loc\sd\msg\3** directory; for custom, use the **\fnsw_loc\sd\msg\100** directory; these match the WEBAPP constants defined in the VWSession API.
2. Select all of the files in the directory, and change the file attributes from Read-only to Read/Write.
3. Locate and make a copy of the appropriate .MSG file. You should change the extension of the copy to .ORG (to indicate it represents the original content of the file).
4. Open the notification file (with the .MSG extension) in a text editor.

5. Modify the HTML content. (Refer to the appropriate notification type topic to determine what system information you can add to the messages.)
6. Save the changes and close the text file.
7. Restart Image Services on the server running Panagon WorkFlo Services. (Perform this step so the **vwnotify** process can cache the changed files when the process restarts.)

Tip Refer to the *Email Notification* section in the *Help for eProcess Administrators* for additional information on configuring email notifications.

eMail Notification Types

The following table lists the email notification types, the file names, and briefly describes the reason why a participant or tracker receives the notification from WorkFlo Services.

Notification type	File name	Description
Step notifications		
Overdue Step Notification	stp_deadline.msg	Sent to a workflow participant to indicate that a step is overdue for completion.
Step Assignment Notification	stp_new.msg	Sent as a notification that the participant was assigned a new step.
Step Reminder Notification	stp_reminder.msg	Sent as a reminder to a workflow participant that a step deadline is approaching.
Tracker notifications		
Tracking Assignment Notification	trk_new.msg	Sent to a participant to indicate that he or she was designated as a workflow tracker.
Overdue Tracker Step Notification	trk_stp_deadline.msg	Sent to a tracker to indicate that a workflow step has not been completed.
Workflow notifications		
Workflow Exception Notification	trk_exception.msg	Sent as a notification to a workflow tracker that a workflow exception occurred.

Workflow Overdue Notification	trk_wf_deadline.msg	Sent to an assigned tracker to indicate that a workflow is overdue for completion.
Workflow Reminder Notification	trk_wf_reminder.msg	Sent to an assigned tracker as a notification that a workflow deadline is approaching.
Milestone notifications		
Workflow Milestone Notification	org_milestone.msg	Sent to the workflow originator, which typically is the person who launched the workflow, as a notification that a workflow milestone was reached.
Workflow Tracker Milestone Notification	trk_milestone.msg	Sent to an assigned tracker as a notification that workflow milestone was reached.

Using Secure Sockets Layer (SSL)

By default, Panagon Web WorkFlo does not use the Secure Sockets Layer (SSL) protocol. Therefore, the email notification files, including the localized versions, use a reference to the web server specified in the vwserver.ini file, and include links similar to:

http://<$F_WEBSERVER>/idmws/. . .

If you enable SSL on your web server, you must modify the http:// protocol designation to https:// for all occurrences of the files you are referencing. For example, all references to a web server would appear similar to the following:

https://<$F_WEBSERVER>/idmws/. . .

Localized Notification Files

Localized versions of the email notification files are in the same folder as the U.S. English files. On the WorkFlo Services server, navigate to the \fnsw_loc\sd\msg\2 directory (for ActiveX). (On UNIX servers, navigate to the /fnsw/loc/sd/msg/2 directory.) Note that for Open Client, go to the \fnsw_loc\sd\msg\3 directory; for custom, use the \fnsw_loc\sd\msg\100 directory; these match the WEBAPP constants defined in the VWSession API.

To activate a localized notification file, rename the U.S. English file name in a manner that will indicate its contents; then rename the localized file to the previous U.S. English file name.

The localized email notification files are named using the U.S. English file name with an appended suffix, which indicates the localized language:

- French (*.msg_fr)
- German (*.msg_de)
- Italian (*.msg_it)
- Spanish (*.msg_es)

The only significant difference in the localized versions of the files and the U.S. English language version is the text displayed in the email notification; none of the variable names are localized.

The following table lists the U.S. English file name and the corresponding localized file name (according to language):

U.S. English	French	German
org_milestone.msg	org_milestone.msg_fr	org_milestone.msg_de
stp_deadline.msg	stp_deadline.msg_fr	stp_deadline.msg_de
stp_new.msg	stp_new.msg_fr	stp_new.msg_de
stp_reminder.msg	stp_reminder.msg_fr	stp_reminder.msg_de
trk_new.msg	trk_new.msg_fr	trk_new.msg_de
trk_stp_deadline.msg	trk_stp_deadline.msg_fr	trk_stp_deadline.msg_de
trk_exception.msg	trk_exception.msg_fr	trk_exception.msg_de
trk_milestone.msg	trk_milestone.msg_fr	trk_milestone.msg_de
trk_wf_deadline.msg	trk_wf_deadline.msg_fr	trk_wf_deadline.msg_de
trk_wf_reminder.msg	trk_wf_reminder.msg_fr	trk_wf_reminder.msg_de

Italian	Spanish
org_milestone.msg_it	org_milestone.msg_es
stp_deadline.msg_it	stp_deadline.msg_es
stp_new.msg_it	stp_new.msg_es
stp_reminder.msg_it	stp_reminder.msg_es
trk_new.msg_it	trk_new.msg_es
trk_stp_deadline.msg_it	trk_stp_deadline.msg_es
trk_exception.msg_it	trk_exception.msg_es
trk_milestone.msg_it	trk_milestone.msg_es
trk_wf_deadline.msg_it	trk_wf_deadline.msg_es
trk_wf_reminder.msg_it	trk_wf_reminder.msg_es

As is the case with the U.S. English email notification files, you can add optional variables in these localized email notification files.

Optional Email Notification Variables

If an optional field is not exposed in the Event Log, that optional variable will be ignored. By default, all of the system fields necessary to supply required variable information are exposed in the Event Log.

Note: While it is technically possible to expose any system field in the event log, it is also possible for a predefined F_xxx system field to be recorded in the event log record incorrectly or meaninglessly. For example, the F_Response field may contain a stale response string.

A user-defined field does not begin with "F_" and it must be exposed in the event log.

A given field variable obtains its values from the field that is named by the string following the "$" prefix in the field variable name, as indicated in the following table.

Field variable	Must be Exposed in Log	Description
$F_EMAILADDR emailAddr		String containing a secondary email address. The mail address specifies additional recipients for the message; separate multiple recipients with a semicolon ("**;**"). Adding the secondary email variable allows the system to send a copy of the notification to a person, or list of people, who would not otherwise be notified. Place the variable outside of the <HTML></HTML> tag set. For example, you might place the second address immediately after the line containing the $F_SUBJPREFIX variable. If you add the variable inside the <HTML></HTML> tag set the additional email addresses are displayed as part of the notification message.
$USERFIELD	yes	String containing information from an exposed user-defined field. "USERFIELD" is actually the name of the user-defined field. You must place the variable inside the <HTML></HTML> tag set. If you place the variable outside the <HTML></HTML> tag set the text is not displayed as part of the notification message.

Note Refer to the *Managing event logs* section in *Events and Statistics* tab of the Configuration Console help for more information on managing or creating exposed system fields.

You can use the non-default variables listed in the following notification files (including the localized versions of these files); for details on available variables, refer to the specific file description.

Overdue Step Notification	stp_deadline.msg
Step Assignment Notification	stp_new.msg
Step Reminder Notification	stp_reminder.msg
Tracking Assignment Notification	trk_new.msg
Overdue Step Notification	trk_stp_deadline.msg
Workflow Exception Notification	trk_exception.msg
Workflow Overdue Notification	trk_wf_deadline.msg
Workflow Reminder Notification	trk_wf_reminder.msg
Workflow Milestone Notification	org_milestone.msg
Workflow Tracker Milestone Notification	trk_milestone.msg

Overdue Step Notification (stp_deadline.msg)

The default **stp_deadline.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the name of the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered by a user when a workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant who started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_EnqueueTime	yes	String containing the time work item entered the queue.
$F_TimeStamp	yes	String containing the time the overdue event occurred.
$F_STEPLABEL		String containing the step label from the workflow definition.
$F_STEPINSTR		String containing the step instructions from the workflow definition.
$F_STEPPROC		String specifying the name of the processor the participant must use to complete the step.

$F_WORKQUEUE		String containing the work queue associated with the step or work item.
$F_WobNum	yes	GUID (Global Unique IDentifier). The GUID identifies the specific work item for the notification

Tip In addition to the default variables listed above, you can add optional variables in the notification file.

When shipped, the stp_deadline.msg file contained the following HTML formatting (the variables appear in bold text):

```
<$F_SUBJPREFIX Overdue: >
<html>
<head>
<title>Overdue Step Notification</title>
</head>
<body background="http://<$F_WEBSERVER>/idmws/images/FN_Logo_Background.gif"
bgcolor="#FFFFFF" link="#FF0000" vlink="#C0C0C0" alink="#FF0000">
<table border="0">
<tr>
<td align="left"><strong>Subject: </strong></td>
<td align="left"><$F_Subject></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspBy: </strong></td>
<td align="left"><$F_Originator[1]></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspOn: </strong></td>
<td align="left"><$F_StartTime></td>
</tr>
<tr>
<td align="left"><strong>Received&nbspOn: </strong></td>
<td align="left"><$F_EnqueueTime></td>
</tr>
<tr>
<td align="left"><strong>Deadline: </strong></td>
<td align="left"><font color="red"><$F_TimeStamp> (OVERDUE!)</font></td>
</tr>
<tr>
<td align="left"><strong>Step&nbspLabel: </strong></td>
<td align="left"><$F_STEPLABEL></td>
</tr>
<tr>
```

```
<td align="left"><strong>Instructions: </strong></td>
<td align="left"><$F_STEPINSTR></td>
</tr>
</table>
<hr>
<table>
<tr>
<td align="center">
<img border="0" src="http://<$F_WEBSERVER>/idmws/images/s_overdue.gif" width="32"
height="32">
</td>
<td>
<a   href="#" OnClick = " window.open ('http://<$F_WEBSERVER>/idmws/PWRedirector.asp?
Redist/WF_Step/< $F_STEPPROC>?
queueName=< $F_WORKQUEUE> &wobId=<$F_WobNum> &stepName=<$F_STEPLABEL>')"
>Step Assignment...</a>
</td>
</tr>
</table>
<hr>
</body>
</html>
```

Step Assignment Notification (stp_new.msg)

The default **stp_new.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
$F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the name of the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. Additionally, the string is used to construct the URL for a step item. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered by the user when a workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant that started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_TimeStamp	yes	String containing the time the step assignment occurred.
$F_STEPLABEL		String containing the step label from the workflow definition.
$F_STEPINSTR		String containing the step instructions from the workflow definition.
$F_STEPPROC		String specifying the processor the participant must use to complete the step.
$F_WORKQUEUE		String containing the work queue associated with the step or work item.
$F_WobNum	yes	GUID (Global Unique IDentifier). The GUID identifies the specific work item for the notification.

Tip In addition to the default variables listed above, you can add optional variables in the notification file.

When shipped, the stp_new.msg file contained the following HTML formatting (the variables appear in bold text):

```
<$F_SUBJPREFIX Work Item: >
<html>
<head>
<title>Step Assignment Notification</title>
</head>
<body background="http://<$F_WEBSERVER>/idmws/images/FN_Logo_Background.gif" bgcolor="#FFFFFF" link="#6666FF"
vlink="#C0C0C0" alink="#6666FF">
<table border="0">
```

```
<tr>
<td align="left"><strong>Subject: </strong></td>
<td align="left"><$F_Subject></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspBy: </strong></td>
<td align="left"><$F_Originator[1]></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspOn: </strong></td>
<td align="left"><$F_StartTime></td>
</tr>
<tr>
<td align="left"><strong>Received&nbspOn: </strong></td>
<td align="left"><$F_TimeStamp></td>
</tr>
<tr>
<td align="left"><strong>Step&nbspLabel: </strong></td>
<td align="left"><$F_STEPLABEL> </td>
</tr>
<tr>
<td align="left"><strong>Instructions: </strong></td>
<td align="left"><$F_STEPINSTR></td>
</tr>
</table>
<hr>
<table>
<tr>
<td align="center">
<img border="0" src="http://<$F_WEBSERVER>/idmws/images/s_step.gif" width="32" height="32">
</td>
<td>
<a  href="#" OnClick = "window.open('http://<$F_WEBSERVER>/idmws/PWRedirector.asp?
Redist/WF_Step/<$F_STEPPROC>?queueName=<$F_WORKQUEUE>&wobId=<$F_WobNum>&stepName=<$F_STEPLABEL>')">Step
Assignment...</a>
</td>
</tr>
</table>
<hr>
</body>
</html>
```

Step Reminder Notification (stp_reminder.msg)

The default **stp_reminder.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
$F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the name of the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered by the user when a workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant that started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_EnqueueTime	yes	String containing the time work item entered the queue.
$F_STEPLABEL		String containing the step label from the workflow definition.
$F_STEPINSTR		String containing the step instructions from the workflow definition.
$F_STEPPROC		String specifying the name of the processor the participant must use to complete the step.
$F_WORKQUEUE		String containing the work queue associated with the step or work item.
$F_WobNum	yes	GUID (Global Unique IDentifier). The GUID identifies the specific work item for the notification.

Tip In addition to the default variables listed above, you can add optional variables in the notification file.

When shipped, the stp_reminder.msg file contained the following HTML formatting (the variables appear in bold text):

```
<$F_SUBJPREFIX Reminder: >
<html>
<head>
<title>Step Reminder Notification</title>
</head>
<body background="http://<$F_WEBSERVER>/idmws/images/FN_Logo_Background.gif" bgcolor="#FFFFFF" link="#6666FF"
vlink="#C0C0C0" alink="#6666FF">
<table border="0">
<tr>
<td align="left"><strong>Subject: </strong></td>
```

```
<td align="left"><$F_Subject></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspBy: </strong></td>
<td align="left"><$F_Originator[1]></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspOn: </strong></td>
<td align="left"><$F_StartTime></td>
</tr>
<tr>
<td align="left"><strong>Received&nbspOn: </strong></td>
<td align="left"><$F_EnqueueTime></td>
</tr>
<tr>
<td align="left"><strong>Step&nbspLabel: </strong></td>
<td align="left"><$F_STEPLABEL> </td>
</tr>
<tr>
<td align="left"><strong>Instructions: </strong></td>
<td align="left"><$F_STEPINSTR></td>
</tr>
</table>
<hr>
<table>
<tr>
<td align="center">
<img border="0" src="http://<$F_WEBSERVER>/idmws/images/s_remind.gif" width="32" height="32">
</td>
<td>
<a   href="#" OnClick = "window.open('http://<$F_WEBSERVER>/idmws/PWRedirector.asp?
Redist/WF_Step/<$F_STEPPROC>?queueName=<$F_WORKQUEUE>&wobId=<$F_WobNum>&stepName=<$F_STEPLABEL>')">Step
Assignment...</a>
</td>
</tr>
</table>
<hr>
</body>
</html>
```

Tracking Assignment Notification (trk_new.msg)

The default **trk_new.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
$F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the name of the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. Additionally, the string is used to construct the URL for a step item. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered by the user when a workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant that started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_WORKFLOW_DEADLINE		String containing a computed value. The value represents the result of the following computation: F_StartTime + F_WFDeadline(*60). (The F_WFDeadline field contains the workflow deadline value, in minutes.)
$F_WORKQUEUE		String containing the work queue associated with the workflow. By default this is the Tracker queue.

$F_WobNum	yes	GUID (Global Unique IDentifier). The GUID identifies the specific work item for the notification. By default this is the work object number for the item in the Tracker queue.

Tip In addition to the default variables listed above, you can add optional variables in the notification file.

When shipped, the trk_new.msg file contained the following HTML formatting (the variables appear in bold text):

```
<$F_SUBJPREFIX Tracking Item: >
<html>
<head>
<title>Tracking Assignment Notification</title>
</head>
<body background="http://<$F_WEBSERVER>/idmws/images/FN_Logo_Background.gif"
bgcolor="#FFFFFF" link="#663399" vlink="#C0C0C0" alink="#663399">
<table border="0">
<tr>
<td align="left"><strong>Subject: </strong></td>
<td align="left"><$F_Subject></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspBy: </strong></td>
<td align="left"><$F_Originator[1]></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspOn: </strong></td>
<td align="left"><$F_StartTime></td>
</tr>
<tr>
<td align="left"><strong>Deadline: </strong></td>
<td align="left"><$F_WORKFLOW_DEADLINE></td>
</tr>
</table>
<hr>
<table>
<tr>
<td align="center">
<img border="0" src="http://<$F_WEBSERVER>/idmws/images/s_track.gif" width="32" height="32">
</td>
<td>
<a href="#" OnClick = "window.open('http://<$F_WEBSERVER>/idmws/PWRedirector.asp?
Redist/WF_Tracker/tracker_main.asp?queueName=<$F_WORKQUEUE>&wobId=<$F_WobNum>')Tracking
Assignment...
</a>
</td>
</tr>
</table>
```

```
<hr>
</body>
</html>
```

Overdue Step Tracker Notification (trk_stp_deadline.msg)

The default **trk_stp_deadline.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
$F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the name of the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered by a user when the workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant who started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_STEPLABEL		String containing the step label from the workflow definition.
$F_BoundUserId[1]	yes	Integer indicating whether the work item is bound to a user. If the work item is bound, lists the user ID of the binding user; 0 means that the work item is not bound (in the work queue).
$F_EnqueueTime	yes	String containing the time work item entered the queue.
$F_TimeStamp	yes	String containing the time the overdue event occurred.

Tip optional variables in the notification file.

When shipped, the trk_stp_deadline.msg file contained the following HTML formatting (the variables appear in bold text):

```
<$F_SUBJPREFIX Overdue Step: >
<html>
<head>
<title>Step Overdue Notification</title>
</head>
<body background="http://<$F_WEBSERVER>/idmws/images/FN_Logo_Background.gif"
bgcolor="#FFFFFF" link="#FF0000" vlink="#C0C0C0" alink="#FF0000">
<table border="0">
<tr>
<td align="left"><strong>Subject: </strong></td>
<td align="left"><$F_Subject></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspBy: </strong></td>
<td align="left"><$F_Originator[1]></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspOn: </strong></td>
<td align="left"><$F_StartTime></td>
</tr>
<tr>
<td align="left"><strong>Step&nbspLabel: </strong></td>
<td align="left"><$F_STEPLABEL> </td>
</tr>
<tr>
<td align="left"><strong>Participant: </strong></td>
<td align="left"><$F_BoundUserId[1]></td>
</tr>
<tr>
<td align="left"><strong>Received&nbspOn: </strong></td>
<td align="left"><$F_EnqueueTime></td>
</tr>
<tr>
<td align="left"><strong>Deadline: </strong></td>
<td align="left"><font color="red"><$F_TimeStamp> (OVERDUE!)</font></td>
</tr>
</table>
<hr>
</body>
</html>
```

Workflow Exception Notification (trk_exception.msg)

The default **trk_exception.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
$F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the name of the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. Additionally, the string is used to construct the URL for a step item. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered when the workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant that started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_TimeStamp	yes	String containing the time the exception occurred.
$F_WORKFLOW_DEADLINE		String containing a computed value. The value represents the result of the following computation: F_StartTime + F_WFDeadline(*60). (The F_WFDeadline field contains the workflow deadline value, in minutes.)

$F_STEPLABEL		String containing the step label from the workflow definition.
$F_STEPINSTR		String containing the step instructions from the workflow definition.
$F_Text	yes	Contains a string of text exception information that was entered by the user who designed the workflow.

Tip In addition to the default variables listed above, you can add optional variables in the notification file.

When shipped, the **trk_exception.msg** file contained the following HTML formatting (the variables appear in bold text):

<**$F_SUBJPREFIX** Workflow Exception: >
<html>
<head>
<title>Workflow Exception Notification</title>
</head>
<body background="http://<**$F_WEBSERVER**>/idmws/images/FN_Logo_Background.gif" bgcolor="#FFFFFF" link="#FF0000" vlink="#C0C0C0" alink="#FF0000">
<table border="0">
<tr>
<td align="left">Subject: </td>
<td align="left"><**$F_Subject**></td>
</tr>
<tr>
<td align="left">Launched By: </td>
<td align="left"><**$F_Originator[1]**></td>
</tr>
<tr>
<td align="left">Launched On: </td>
<td align="left"><**$F_StartTime**></td>
</tr>
<tr>
<td align="left">Received On: </td>
<td align="left"><**$F_TimeStamp**></td>
</tr>
<tr>
<td align="left">Deadline: </td>
<td align="left"><**$F_WORKFLOW_DEADLINE**></td>
</tr>
<tr>
<td align="left">Step Label: </td>

```
<td align="left"> <$F_STEPLABEL> </td>
</tr>
<tr>
<td align="left"><strong>Instructions: </strong></td>
<td align="left"> <$F_STEPINSTR></td>
</tr>
</table>
<hr>
<p>
<font color="#FF0000">EXCEPTION <$F_Text></font>
</p>
<hr>
</body>
</html>
```

Workflow Overdue Notification (trk_wf_deadline.msg)

The default **trk_wf_deadline.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
$F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered by the user when a workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant that started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_WORKFLOW_DEADLINE		String containing a computed value. The value represents the result of the following computation: F_StartTime + F_WFDeadline(*60). (The F_WFDeadline field contains the workflow deadline value, in minutes.)
$F_WORKQUEUE		String containing the work queue associated with the work flow. By default this is the Tracker queue.
$F_WobNum	yes	GUID (Global Unique IDentifier). The GUID identifies the specific work item for the notification. By default this is the work object number for the item in the Tracker queue.

Tip In addition to the default variables listed above, you can add optional variables in the notification file.

When shipped, the trk_wf_deadline.msg file contained the following HTML formatting (the variables appear in bold text):

```
<$F_SUBJPREFIX Overdue Workflow: >
<html>
<head>
<title>Workflow Overdue Notification</title>
</head>
<body background="http://<$F_WEBSERVER>/idmws/images/FN_Logo_Background.gif"
bgcolor="#FFFFFF" link="#FF0000" vlink="#C0C0C0" alink="#FF0000">
<table border="0">
<tr>
<td align="left"><strong>Subject: </strong></td>
<td align="left"><$F_Subject></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspBy: </strong></td>
<td align="left"><$F_Originator[1]></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspOn: </strong></td>
<td align="left"><$F_StartTime></td>
</tr>
<tr>
<td align="left"><strong>Deadline: </strong></td>
<td align="left"><font color="red"><$F_WORKFLOW_DEADLINE> (OVERDUE!)</font></td>
</tr>
</table>
<hr>
<table>
<tr>
<td align="center">
<img border="0" src="http://<$F_WEBSERVER>/idmws/images/s_overdue.gif" width="32" height="32">
</td>
<td>
<a href="#" OnClick = "window.open('http://<$F_WEBSERVER>/idmws/PWRedirector.asp?
Redist/WF_Tracker/tracker_main.asp?queueName=<$F_WORKQUEUE>&wobId=<$F_WobNum>')"Tracking
Assignment...
</a>
</td>
</tr>
</table>
<hr>
</body>
</html>
```

Workflow Reminder Notification (trk_wf_reminder.msg)

The default **trk_wf_reminder.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
$F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the name of the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered by the user when a workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant that started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_WORKFLOW_DEADLINE		String containing a computed value. The value represents the result of the following computation: F_StartTime + F_WFDeadline(*60). (The F_WFDeadline field contains the workflow deadline value, in minutes.)
$F_WORKQUEUE		String containing the work queue associated with the workflow. By default this is the Tracker queue.
$F_WobNum	yes	GUID (Global Unique IDentifier). The GUID identifies the specific work item for the notification. By default this is the work object number for the item in the Tracker queue.

Tip In addition to the default variables listed above, you can add optional variables in the notification file.

When shipped, the **trk_wf_reminder.msg** file contained the following HTML formatting (the variables appear in bold text):

```
<$F_SUBJPREFIX Workflow Reminder: >
<html>
<head>
<title>Workflow Reminder Notification</title>
</head>
<body background="http://<$F_WEBSERVER>/idmws/images/FN_Logo_Background.gif" bgcolor="#FFFFFF"
link="#6666FF" vlink="#C0C0C0" alink="#6666FF">
<table border="0">
<tr>
<td align="left"><strong>Subject: </strong></td>
<td align="left"><$F_Subject></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspBy: </strong></td>
<td align="left"><$F_Originator[1]></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspOn: </strong></td>
<td align="left"><$F_StartTime></td>
</tr>
<tr>
<td align="left"><strong>Deadline: </strong></td>
<td align="left"><$F_WORKFLOW_DEADLINE></td>
</tr>
</table>
<hr>
<table>
<tr>
<td align="center">
<img border="0" src="http://<$F_WEBSERVER>/idmws/images/s_remind.gif" width="32" height="32">
</td>
<td>
<a href="#" OnClick = "window.open('http://<$F_WEBSERVER>/idmws/PWRedirector.asp?
Redist/WF_Tracker/tracker_main.asp?queueName=<$F_WORKQUEUE>&wobId=<$F_WobNum>')">Tracking
Assignment...
</a>
</td>
</tr>
</table>
<hr>
</body>
</html>
```

Workflow Milestone Notification (org_milestone.msg)

The default **org_milestone.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
$F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the name of the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. Additionally, the string is used to construct the URL for a step item. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered by a user when a workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant who started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_MILESTONE_NAME		String containing the name of the milestone, as it was defined in Designer, that was reached.
$F_Text	yes	String containing the text entered in the **Message** field, in Designer, when the the milestone was defined. Typically the string contains additional information about what the milestone notification means to the user who received it.
$F_WorkSpaceId	yes	String containing the workspace ID value assigned when the workflow was first transferred.

$F_WorkClassId	yes	String containing a representation of the workflow number.
$F_WorkFlowNumber	yes	GUID (Global Unique IDentifier). The GUID identifies the unique instance of the active workflow.

Tip In addition to the default variables listed above, you can add optional variables in the notification file.

The JavaScript included in the message constructs a unique request for the VWProcess object necessary to fetch the milestone array containing this milestone. The script passes the exposed field data to the milestone.asp file, on the Web WorkFlo server, to display a list of the milestones.

When shipped, the org_milestone.msg file contained the following HTML formatting (the variables appear in bold text):

```
<$F_SUBJPREFIX Workflow Milestone Reached: >
<html>
<head>
<title>Workflow Milestone Notification</title>
<SCRIPT language="JavaScript"
src="http://<$F_WEBSERVER>/idmws/Redist/WF_Html_Toolkit/Utility/utility.js"></script>

<SCRIPT language="JavaScript">
<!--
function onOpenMilestones(webServer, workSpaceID, workClassId, workflowNumber, subject,
launchDate)
{
var subjectURL = convertToSpecialChar(subject);
var launchDateURL = convertToSpecialChar(launchDate);

window.open("http://" + webServer
+"/idmws/PWRedirector.asp?Redist/WF_Milestone/milestone.asp?workSpaceId=" + workSpaceID
+"&workClassId=" + workClassId + "&workflowNumber=" + workflowNumber + "&subject="
+subjectURL +"&launchDate=" + launchDateURL);
}
//-->
</SCRIPT>
</head>
<body background="http://<$F_WEBSERVER>/idmws/images/FN_Logo_Background.gif"
bgcolor="#FFFFFF" link="#6666FF" vlink="#C0C0C0" alink="#6666FF">
<table border="0">
<tr>
<td align="left"><strong>Subject: </strong></td>
<td align="left"><$F_Subject></td>
</tr>
```

```
<tr>
<td align="left"><strong>Launched&nbspBy: </strong></td>
<td align="left"><$F_Originator[1]></td>
</tr>
<tr>
<td align="left"><strong>Launched&nbspOn: </strong></td>
<td align="left"><$F_StartTime></td>
</tr>
</table>
<hr>
The Milestone, <$F_MILESTONE_NAME>, has been reached with the following message: <$F_Text>
<hr>
<table>
<tr>
<td align="center">
<img border="0" src="http://<$F_WEBSERVER>/idmws/images/milestone.gif" width="32"
height="32">
</td>
<td>
<a  href="#" OnClick="JavaScript:onOpenMilestones('<$F_WEBSERVER>', '<$F_WorkSpaceId>',
'<$F_WorkClassId>', '<$F_WorkFlowNumber>', <$F_Subject>, <$F_StartTime>)">Milestone
Tracking...</a>
</td>
</tr>
</table>
</body>
</html>
```

Workflow Tracker Milestone Notification (trk_milestone.msg)

The default **trk_milestone.msg** notification file uses the following WorkFlo Services variables. Some notification information is taken from the exposed log fields; if you disable the fields, you will not be able to use the field values in the notification. All field variables are case sensitive.

Field variable	Exposed	Description
$F_SUBJPREFIX		String containing the prefix for the subject line of the notification message.
$F_WEBSERVER		String containing the name of the Web WorkFlo server a participant must attach to in order to respond to the notice. The name is also used to supply the background and notification type images. The web server value is taken from the **vwserver.ini** file on the WorkFlo Services server.
$F_Subject	yes	String containing the subject entered by the user when a workflow was launched.
$F_Originator[1]	yes	Integer containing the user ID of the participant that started the workflow. Including [1] causes the Image Services user name to display in the notification.
$F_StartTime	yes	String containing the time the workflow was created. With the exception of the initial work item of the workflow, it is different than the $F_CreateTime value.
$F_WORKFLOW_DEADLINE		String containing a computed value. The value represents the result of the following computation: F_StartTime + F_WFDeadline(*60). (The F_WFDeadline field contains the workflow deadline value, in minutes.)
$F_MILESTONE_NAME		String containing the name of the milestone, as it was defined in Designer, that was reached.

$F_Text	yes	String containing the text entered in the **Message** field, in Designer, when the the milestone was defined. Typically the string contains additional information about what the milestone notification means to the user who received it.

Tip In addition to the default variables listed above, you can add optional variables in the notification file.

When shipped, the trk_milestone.msg file contained the following HTML formatting (the variables appear in bold text):

<**$F_SUBJPREFIX** Workflow Milestone Reached: >
<html>
<head>
<title>Workflow Tracker Milestone Notification</title>
</head>
<body background="http://<**$F_WEBSERVER**>/idmws/images/FN_Logo_Background.gif" bgcolor="#FFFFFF" link="#6666FF" vlink="#C0C0C0" alink="#6666FF">
<table border="0">
<tr>
<td align="left">Subject: </td>
<td align="left"><**$F_Subject**></td>
</tr>
<tr>
<td align="left">Launched By: </td>
<td align="left"><**$F_Originator[1]**></td>
</tr>
<tr>
<td align="left">Launched On: </td>
<td align="left"><**$F_StartTime**></td>
</tr>
<tr>
<td align="left">Deadline: </td>
<td align="left"><**$F_WORKFLOW_DEADLINE**></td>
</tr>
</table>
<hr>
The Milestone, <**$F_MILESTONE_NAME**>, has been reached with the following message:
<$F_Text>
<hr>
</body>
</html>

Set up a SMTP Server

This topic describes the general steps you must take to set up a SMTP server for testing your eProcess Services Notifications. Since there are many commercially available SMTP applications and hundreds of different configurations, you should view the procedures listed as general guidelines. You can set up SMTP services on the Web WorkFlo server; however, you must also install a SMTP-compliant mail server (like Microsoft Exchange) on the same server or have access to a server with the mail server application already installed.

Note Regardless of the mail server application you choose, you must refer to the installation and administration documentation for that mail server software to answer specific configuration or user interface questions.

Configure SMTP Services on Windows 2000

The SMTP Service is installed with the Windows 2000 operating system. Perform the following steps to configure the SMTP Service on your server:

1. On your designated server, point to Start. Navigate to Programs, point to Administrative Tools, then point to and click Internet Service Manager. The Microsoft Management Console (MMC) displays.
2. From MMC, locate and expand the Internet Information Services. Locate and highlight your server's name. Expand the options under your server's name.
3. Locate the Default SMTP Virtual Site option, and expand it. From Domains create a remote domain. Specify a unique domain name (for example, mailtest.com). Make the name unique so there are no conflicts across the current domain.
4. Highlight and click on your new domain. Right-click and choose Properties. Select Allow incoming mail to be relayed to this domain option. Leave all other settings as the default. Click OK. Exit MMC.

Configure SMTP Services on Windows NT

In order to perform the procedure listed in this section, you must have already installed the SMTP Services from the Windows NT 4.0 Option Pack installer. Perform the following steps to configure the SMTP Service on your server:

1. On your designated server, point to Start. Navigate to Programs, point to Windows NT 4.0 Option Pack, then point to Microsoft Internet Information Server. Finally, point to and click Internet Service Manager. The Microsoft Management Console (MMC) displays.
2. From MMC, locate and expand the Internet Information Server. Locate and highlight your server's name. Expand the options under your server's name.
3. Locate the Default SMTP Site option, and expand it. From Domains create a remote domain. Specify a unique domain name (for example, mailtest.com). Make the name unique so there are no conflicts across the current domain.

4. Highlight and click on your new domain. Right-click and choose Properties. Select Allow incoming mail to be relayed to this domain option. Leave all other settings as the default. Click OK. Exit MMC.

Install and configure a mail services application

Install a preferred mail services application (for example, Microsoft Exchange or ArGoSoft Mail Server). You can download an evaluation copy of the ArGoSoft Mail Server (http://www.argosoft.com), which is a SMTP/POP3/Finger capable application, for testing purposes.

While installing and configuring your mail server software, you will have to supply the following information (not necessarily in this order):

- Domain name you specified in the SMTP Service properties dialog
- Number of mailboxes being used
- Image Services and/or Content Services user names (when creating mail user accounts)

Configure the mail client

This procedure assume you are using Microsoft Outlook as a mail client. Perform the following general steps to configure Microsoft Outlook as your mail client:

1. On a client, point to Start. Navigate to Settings, and point to and click Control Panel.
2. From the Control Panel window, open Mail.
3. Click Show Profiles and click Add.
4. From the Microsoft Outlook Setup Wizard dialog box, select Internet E-mail only. Click Next.
5. Add a user profile name (for example, SysAdmin). Click Next. Click Setup Mail Account.
6. In the General tab, enter an Account Name (for example, "Primary test mail account"). Configure other account information as needed.
7. In the Servers tab, verify or enter the name of the Incoming Mail (POP3) Server, which should be the name of the server running your mail server software. Verify or enter the Outgoing mail server (SMTP), which should be the name of the server with both IIS and SMTP running.
8. In the Incoming Mail Server Account Name field, enter the user name and password you created during the mail server configuration stage. Click OK.
9. From the Microsoft Outlook Setup Wizard dialog box, click Next for Internet Email. Enter a name for the Personal Address Book.
10. Enter a similar name for the Personal Folders option. Click Next.
11. Repeat these steps for any other users you created.

Configure eProcess email notification

On your Panagon WorkFlo Server, edit the vwserver.ini file, which resides in the \fnsw_loc\sd directory, to include the SMTP Server name and the Web WorkFlo server name. Your file might contain entries similar to the following:

SMTPHost=CRONOS
EmailLogonId=SysAdmin@mailtest.com
EmailFromId=SysAdmin@mailtest.com
WebServer=CRONOS

where CRONOS is the name of the server running SMTP Services and Web WorkFlo, and SysAdmin@mailtest.com is the user name created during your mail server configuration.

Configure your TCP/IP connections

Modify the HOSTS file, on every computer you are using for testing, to include all servers needed to participate in the workflow. For example, include the address and names of the system running the mail server application, the web server, and WorkFlo Services server. A generalized HOSTS file might appear as shown below:

127.0.0.1 localhost
10.0.1.101 cronos #Web WorkFlo server
10.0.1.242 titan #mail server
10.0.1.451 warbird #WorkFlo Services server

You need not perform this step if your servers are enlisted in a DHCP configuration.

Using ADO to Query the eProcess Database

Querying eProcess queues, rosters, or logs is a typical action performed by a Step Processor (or "work performer") application. These queries normally use appropriate eProcess APIs to retrieve the data. While the application queries a queue, roster, or log, it is understood that the data retrieved by these queries actually resides in the underlying eProcess database tables (either a SQL Server or Oracle database). However, in certain instances, for example, where the query is to retrieve a large amount of data, performance can become an issue (for details, see Why use ADO? — Query Performance Issues below).

In these cases, an alternative approach that can improve performance, is to bypass the API and query the eProcess database views directly, using an Active Directory Object (ADO) Connection object or ODBC to directly connect to the database. While there is no requirement for using this approach (using the eProcess API classes and methods is the standard way to do queries), the ADO approach can significantly improve performance when you need to query large numbers of data records.

This topic describes how to use ADO to query the eProcess database, as follows:

Note Use of this approach requires an understanding of the eProcess database schema, ADO, and some SQL.

ADO eProcess Query Overview

The basic premise of the ADO approach is that instead of querying on a queue, roster, or log via the eProcess APIs, you instead query on database views generated by the eProcess server. These views are of the underlying eProcess database tables associated with the queues, rosters, and logs of a given isolated region. Using the ADO approach, you can directly query on exposed fields of these database views via an ADO Connection object, completely bypassing the eProcess API. For a step-by-step procedure on how to do this, see Using the ADO Query Procedure below.

A typical use case for the ADO query approach involves situations where a user application needs to query for a large amount of data. In this case, you might retrieve the data records with an ADO query, then use the data items retrieved to query specific work items in eProcess. For example, you might use the `F_WobNum` field retrieved as part of the data record to query for a specific work item using the eProcess API.

To accomplish these steps involves an understanding of the following issues:

- Understanding eProcess database views
- Why use ADO? — Query performance issues
- Fields available for queries

Caution It is strongly recommended that you not proceed without first reviewing these issues. In addition, under no circumstances should you attempt to directly modify the eProcess database.

Back to top

Understanding eProcess Database Views

Each time a user performs a commit operation on configuration changes on the workflow server, or a transfer on a workflow definition, the eProcess server automatically creates database views of those database tables related to the isolated region's queues, rosters, and logs for the workflow definition. This occurs automatically, with no customization required (or recommended). The existence of these database views enables exposed fields associated with the tables in these views to be directly queried from your application or applet via ADO or ODBC (not all exposed fields, only those useful in a practical way for queries are actually available — for additional information, see Fields available for queries below).

The following naming conventions are used for the database views:

- **Queue database view:** VWVQ<*isolated region*>_<*workflow definition queue name*>; for example, for isolated region 21 and the Inbox queue: `VWVQ21_Inbox`
- **Roster database view:** VWVR<*isolated region*>_<*workflow definition roster name*>; for example, for isolated region 21 and the DefaultRoster view: `VWVR21_DefaultRoster`
- **Log database view:** VWVL<*isolated region*>_<*workflow definition queue name*>; for example, for isolated region 21 and the mylog view: `VWVQ21_mylog`

It is important to note that the read/write permissions associated with these database views created by the eProcess server vary according to whether the underlying database is Oracle or MS SQL Server. The Oracle database views created by the eProcess server are read-only. MS SQL Server data views are read/write (SQL Server does not provide a way to create read-only views.)

Warning If you are accessing a SQL Server database, it is essential that you do not directly modify the database directly, even though the DBMS allows it. If you do modify a SQL Server database directly, eProcess processing will not work correctly. It is recommended that the Database Administrator make the underlying tables (on which the database views are based) Read-only for users.

Back to top

Why Use ADO? — Query Performance Issues

eProcess query performance, is affected by a number of factors. The most important of these factors are:

- **The types of objects returned by the query:** Normal API eProcess queries can return any of three types of objects: data records (queue element and roster element objects), step element objects, and work objects. Of these, the most expensive in terms of performance, are work objects, because they require deblobbing. Next most expensive are step elements, that require some deblobbing. Data records are the least expensive objects, requiring no deblobbing. In addition, ADO queries are more efficient because they return only a subset of the information retrieved by API queries. In particular, ADO queries do not return additional isolated region data that is returned by API queries.
- **The data transformations:** The number and types of data transformations between the client and the database can also affect performance. For example, with an eProcess API query, there is the data transformation between the SQL data records returned by the DBMS and the C++ data objects handled by the eProcess server, then the data transformation between C++ and the Java objects used by the eProcess Java API. If the Windows client application is non-Java, an additional transformation between Java and COM (via JiGlue) is also required. On the other hand (regardless of the source language you write your application in), you specify the ADO query using SQL, thereby reducing the number of data transformations needed.
- **The use of indexes:** Although the use of indexes is recommended for the eProcess API, some developers may not use indexes for their custom application or processor queries. As a result, query data retrieval is less efficient, particularly if they need to retrieve large amounts of data from the database. The developer (or the

Database Administrator) can add indexes to the database to additionally improve performance.

Back to top

Fields Available for Queries

You query on exposed fields provided in the database view. Therefore, each of the fields that you need to query must exist as an exposed field on a queue, roster, or log, as defined by a workflow definition. Fields that are not exposed are not included in the database view. Of all the exposed fields, those exposed system fields that are not useful for ADO queries are automatically excluded from the view. Similarly, large binary fields and BLOBS are also not included.

In other words, only those exposed fields that are useful in a practical way for ADO queries are actually available.

The fields in the database views have the same names as the exposed fields in the underlying database tables from which they are derived. However, the following database fields are not allowed in the database views:

- Fields that contain database reserved words.
- Fields whose names may present conflict with eProcess server software.
- Fields whose names are longer than 32 characters.
- Field names that differ from the names of their respective fields in the underlying database.

Back to top

Before You Start — Things to Keep in Mind

Before you start using ADO queries, you should keep the following important guidelines and limitations in mind:

- **Reminder**: ADO queries are based on database views. Database records should not be updated via ADO.
- Each client that is to use ADO eProcess queries must be individually configured.
- The database client and the database server (whether using Oracle or SQL Server) cannot be on the same machine.
- If you are using an Oracle database, Listener software must be set up first on the server. Then the Oracle network client must be set up before you can configure the ADO/ODBC connection.
- As indicated by the previous **Warning**, if you are using an MS SQL Server database, since SQL Server does not provide a way to create read-only views, it is essential that you do not modify the SQL Server database. If you do so, eProcess processing will not work correctly.

Back to top

Using the ADO Query Procedure

To use ADO or ODBC to query the eProcess database views for queue, roster, and log data, requires the following basic steps, which are performed in the order shown:

Note This procedure varies somewhat, depending upon whether the underlying database is Oracle or MS SQL Server.

1. Configuring the DBMS server (Oracle only).
2. Configuring the client to connect to the database (Oracle only).
3. Configuring an ADO connection (Oracle and MS SQL Server)

4. Identifying the queues, rosters, and logs (and their associated database views).
5. Specifying the query.

These steps are described in detail below.

Configuring the DBMS Server (Oracle only)

This step describes how to configure the Oracle server for ADO eProcess queries. If you are using a SQL Server database, skip to the step, Configuring an ADO Connection.

Caution The database client and the database server cannot be on the same machine. (The Oracle database server and the eProcess server can, at your option, be on the same machine.)

To configure an Oracle server for ADO, use the following procedure:

1. Logon to the Oracle DBMS server as the `fnsw` user.
2. cd to $ORACLE_HOME. Verify that you are logged on as the `fnsw` user.
3. Type **netca** at the command line to bring up the Oracle Net(work) Configuration Assistant.
4. Using the Net Configuration Assistant, install and configure Oracle network configuration on the Oracle server by configuring (in the order shown — if you make a mistake, the Assistant allows you to backtrack):
    - Listener configuration
    - Naming methods configuration
    - Local net service name configuration

    Start by selecting Listener Configuration and click Next.
5. Select Add (to add a Listener) and click Next.
6. Enter the Listener name (for example, `LISTENER`) into the Listener Name field and click Next.
7. Select the network protocol (for example, `TCP`) and click Next. If you are using TCP/IP, select the standard port number of 1521 and click Next.
8. Select No when prompted to configure another Listener and click Next. Listener configuration is complete.
9. Now proceed to net service naming configuration. Select Naming Methods configuration and click Next.
10. Select `Local` as the net service naming method (to understand why, see the next step) and click Next. Naming methods configuration is complete.
11. Select Local Net Service Name configuration (to enable you to use a local name for the eProcess server to access the Oracle database using net service) and click Next.
12. Select Add (to specify you want to add a local name) and click Next.
13. Select the version of the Oracle DBMS (Oracle 8i) and click Next. Then enter the Oracle database System Identifier (SID), for example, IDB, into the Database SID field.
14. Select the network protocol to use to access the Oracle database (for example, `TCP`) and click Next.
15. Enter the eProcess server host name (for example, `eprocess_svr`) into the Host Name field. If you are using TCP/IP, select the standard port number of 1521 and click Next.
16. Select No, do not test, to specify that you don't want to test the database connection at this time (you will do so later in the procedure) and click Next.
17. Enter the net service name (for example, IDBGlobal) into the Net Service Name field and click Next.
18. Select No when prompted to configure another net service name and click Next. Net service name configuration is complete. Click Finish to complete Oracle network setup and exit the Net Configuration Assistant.

19. Type **netasst** at the command line to configure the Oracle network service you just created. The Oracle Net8 Assistant displays.
20. In the Explorer-like pane, under Net8 Configuration, select Local > Listeners > LISTENER. In the main pane, select Database Services. Edit the fields using the entries you specified while creating the net service name (for example, `IDBGlobal` in the Global Database Name field, `IDB` in the SID field). Specify the Oracle root directory (for example, `/usr/ora/817`) in the Oracle Home Directory field. Then select Save from the File menu to save the network configuration.
21. To verify that the Listener has been successfully created, type **Isnrctl** at the command line. Then type **status**. Verify that the SID you specified (for example, `IDB`) is listed as a Listener. If yes, go to the next step. If not, type **stop**, then type **exit**. Restart the Listener (by typing **start** and pressing Enter). Type **Isnrctl** at the command line, then type **status** to verify again that the SID specified is listed as a Listener.
22. You can verify whether the connection to the Oracle database is working using the Oracle Net8 Assistant or manually (see next step). To verify that the Oracle database connection is working using the Net8 Assistant, in the Explorer-like pane, under Net8 Configuration, select Local > Service Naming, then select your net service name (for example, `IDBGlobal`). To test the connection, select Test Service from the Command menu.
    - If the test is successful, the Connection Test dialog displays a message similar to:

      ```
      Attempting to connect using userid: system
      The connection test was successful
      ```

      If successful, close the Connection Test dialog and exit the Net8 Assistant.
    - If the test is not successful, the Connection Test dialog displays a message similar to:

      ```
      The test did not succeed.
      ORA-01017: Invalid username/password; logon denied
      There may be an error in the fields entered,
      or the server may not be ready for a connection.
      ```

      If the test is not successful, click the Change Login button. In the dialog, change the login to username=system and password=manager. Click OK and rerun the test by click the Test button on the Connection Test dialog.
23. To manually test whether the connection to the Oracle database is working, type **sqlplus system/manager@idbglobal** at the command line (where idbglobal is the SID name). This starts SQL *Plus. Perform a simple SQL query. For example, specify: **show parameters db_name** to verify that you are connecting to the correct database.

Back to top

Configuring the Client to Connect to the Database (Oracle Only)

This step describes how to configure an Oracle client to connect to the eProcess database. If you are using a SQL Server database, skip to the step, Specifying an ADO Connection.

This procedure assumes that you have already installed the Oracle 8i client on a Windows machine. Each client that is to use ADO eProcess queries must be individually configured.

Caution The database client and the database server cannot be on the same machine.

Note Oracle 8i is the only Oracle client currently supported. If you install the Oracle 8i client on a Windows 2000 machine, be sure to run the Oracle 8i patches that allow the Oracle Installer to run on Windows 2000 before you start Oracle 8i client installation.

1. Click on Start and select Programs > Oracle - OraHome81 to start the Oracle client.

2. Select Network Administration > Launch Net8 Configuration Assistant to bring up the Oracle Net(work) Configuration Assistant.
3. Using the Net Configuration Assistant, configure the net service on the client by configuring (in the order shown — if you make a mistake, the Assistant allows you to backtrack):
   - Naming methods configuration
   - Local net service name configuration

   Start by selecting Naming Methods Configuration and click Next.
4. Select `Local` as the net service naming method and click Next. Naming methods configuration is complete.
5. Select Local Net Service Name configuration (to enable you to use a local name for the eProcess server to access the Oracle database using net service) and click Next.
6. Select Add (to specify you want to add a local name) and click Next.
7. Select the version of the Oracle DBMS (Oracle 8i) and click Next, then enter the Oracle database System Identifier (SID), for example, IDB, into the Database SID field.
8. Select the network protocol to use to access the Oracle database (for example, `TCP`) and click Next.
9. Enter the eProcess server host name (for example, `eprocess_svr`) into the Host Name field. If you are using TCP/IP, select the standard port number of 1521 and click Next.
10. Select Yes, perform a test, and click Next, to have the Net8 Configuration Assistant test whether database connection is working. (You can also test the connection manually — see step 13.)

    If the test is successful, the Connection Test dialog displays a message similar to:

    ```
    Connecting...Test successful.
    ```

    If successful, click Next.

    - If the test is not successful, the Connection Test dialog displays a message similar to:

      ```
      Connecting...ORA-01017: Invalid username/password; logon denied
      The test did not succeed.

      Some of the information you provided may be incorrect.
      Press Back to review the information provided for net service
      name, or Change Login to change username.
      ```

      If the test is not successful, click the Change Login button. In the dialog, change the login to username=`system` and password=`manager` and click OK to rerun the test.
11. Enter the net service name (for example, IDBGlobal) into the Net Service Name field and click Next.
12. Select No when prompted to configure another net service name and click Next. Net service name configuration is complete. Click Finish to complete Oracle network setup and exit the Net Configuration Assistant.
13. To manually test whether the connection to the Oracle database is working, type **sqlplus system/manager@idbglobal** at the command line (where idbglobal is the SID name). This starts SQL *Plus. Perform a simple SQL query. For example, specify: **show parameters db_name** to verify that you are connecting to the correct database.

    Alternatively, you can also verify that the database connection is working by using the SQL *PLUS utility from the Start menu and entering idbglobal as the Host String (where idbglobal is the SID name).

Back to top

Configuring an ADO Connection on the Client

This step describes how to configure an ODBC data source on the client for ADO eProcess queries (assumes a Windows client platform). Each client that is to use ADO eProcess queries must be individually configured to specify the driver, the Data Source Name (DSN) and the net service name you created previously. The procedure varies, depending upon whether you are configuring for either an Oracle or a MS SQL Server database, as follows:

- Specifying ADO ODBC for Oracle
- Specifying ADO ODBC for MS SQL Server

Specifying ADO ODBC for Oracle

To set up an ADO ODBC connection for an Oracle eProcess database, use the following procedure (the procedure varies somewhat, depending upon whether you use the Oracle ODBC driver or the Microsoft ODBC for Oracle driver):

1. Click on Start and select Settings > Control Panel > Administrative Tools. Double-click on Data Sources (ODBC) to open the ODBC Data Source Administrator dialog. Click on the System DSN tab, then click the Add button.
2. Select either of the following drivers for Oracle (both are supported) in the Create New Data Source dialog.
   - Oracle ODBC Driver — if you use this driver, go to the next step (Step 3).
   - Microsoft ODBC for Oracle — if you use this driver, go to Step 4.

   Then click Finish.
3. On the Oracle8 ODBC Driver Setup dialog:
   - In the Data Source Name field, enter the DSN (for example, eprocess_svr_dsn).
   - In the Description field, enter an appropriate description (for example, eProcess data source name).
   - In the Service Name field, enter the net service name you created previously (for example, IDBGlobal). You must use the same net service name that you configured for this client.
   - You may leave the UserID field blank or enter a userid, as is appropriate.
   - Check Application Options as appropriate.

   Once you have specified the configuration, click OK. You are returned to the ODBC Data Source Administrator dialog, where you should see the new DSN you just configured. Click OK. ODBC setup using the Oracle ODBC driver is complete.
4. On the Microsoft ODBC for Oracle Setup dialog:
   - In the Data Source Name field, enter the DSN (for example, process_svr_dsn).
   - In the Description field, enter an appropriate description (for example, eProcess data source name).
   - You may leave the UserID field blank or enter a userid, as is appropriate.
   - In the Server field, enter the net service name you created previously (for example, IDBGlobal). You must use the same net service name that you configured for this client.

   Once you have specified the configuration, click OK. You are returned to the ODBC Data Source Administrator dialog, where you should see the new DSN you just configured. Click OK. ODBC setup using the Microsoft ODBC driver for Oracle is complete.

Specifying ADO ODBC for MS SQL Server

To set up and test an ADO ODBC connection for a SQL Server eProcess database, use the following procedure:

Note If you are using SQL Server as the DBMS for the WorkFlo server, no additional configuration is needed on the server.

1. Click on Start and select Settings > Control Panel > Administrative Tools. Double-click on Data Sources (ODBC) to open the ODBC Data Source Administrator dialog. Click on the System DSN tab, then click the Add button.
2. Select the SQL Server in the Create New Data Source dialog and click Finish.
3. On the Create a New Data Source to SQL Server dialog:
    - In the Name field, enter the DSN (for example, eprocess_svr_dsn).
    - In the Description field, enter an appropriate description (for example, eProcess data source name).
    - In the Server field (in response to the question: `Which SQL Server do you want to connect to?`), enter the net service name you created previously (for example, IDBGlobal). You must use the same net service name that you configured for this client.
    - Click Next.
    - Specify the authentication method. If you specify using: `With SQL Server authentication using a login ID and password entered by the user`, then:
        - Check the Connect to SQL Server to obtain default settings for the additional configuration options.
        - Specify the DSN login ID and password.
        - Click Next.
    - Check the `Change the default database to` checkbox and enter enter the database name as the default (for example, appsdb). It is recommended that you also check the `Use ANSI quoted identifiers` and `Use ANSI nulls, paddings and warnings` checkboxes. Click Next.
    - The default SQL Server system messages is English.
    - Check the Perform translation for character data checkbox.
    - The default query log file is C:\QUERY.LOG and the default Long query time is 30,000 milliseconds.
    - The default ODBC driver statistics log file is: C:\STATS.LOG.
    - Click Finish to complete setup.
4. The ODBC Microsoft SQL Server Setup dialog displays the SQL Server ODBC driver version and the settings you specified for the data source you just created.
5. To test the connection, click Test Data Source. If the test is successful, the following message is displayed:

    TESTS COMPLETED SUCCESSFULLY!

    Click OK.
6. You are returned to the ODBC Data Source Administrator dialog, where you should see the new DSN you just configured. Click OK. ODBC setup for Microsoft SQL Server is complete.

Back to top

Identifying Queues, Rosters, and Logs

In order to query a database view, you will need to identify the database tables associated with the queues, rosters and logs within a given isolated region. To identify the appropriate table views:

- Use your DBMS and its appropriate tool.
- Remember the naming conventions for eProcess database views (see Understanding eProcess Database Views above).

Back to top

Specifying the Query

You can perform an ADO query by specifying a SQL query in your application. The SQL SELECT statement should reference the database views (of a queue, roster, or log) rather than the physical database tables. The database views were identified in the previous step, Identifying Queues, Rosters, and Logs.

When selecting from VW database views, it is normally necessary to use quotation marks around column names (the column names in the view definition are often defined using quoted identifiers). This is necessary because a user-defined column may actually be a SQL reserved word (e.g., "SELECT" or "FROM"). Or the user-definied column name may not be unique unless the name is defined in a case-sensitive way (i.e., "name" and "Name"). Quoting a column name in a SQL query indicates to the DBMS that the quoted name is a column name, even if it happens to be an otherwise Reserved word. It also indicates to the DBMS that the name is case-sensitive.

When the view being queried is for a queue or roster, you can omit the quotation marks around the VW Server field names (the ones that begin with "F_"). However, all user-defined column names should be quoted.

For example:

```
select F_WobNum, F_WorkSpaceId, "userfieldname1", "userfieldname2"
from f_sw.VWVQ94_testmodswp;
```

Note that `f_sw` identifies the eProcess database.

When the view being queried is for a log, all column names should be quoted, including the VW Server field names. For example:

```
select "F_WobNum", "F_WorkSpaceId", "userfieldname1",
"userfieldname2" from f_sw.vwvl94_testparentlog;
```

For an additional example of how the query is used, see ADO eProcess Query Example Code below.

Back to top

ADO eProcess Query Example Code

The Visual Basic code fragment example below illustrates the use of the ADO query based on database views created on eProcess tables. In this example, the ADO Connection object, ADODB.Connection, is set and the SQL SELECT statement queries for all records for the Inbox queue of isolated region 262.

```
Private Sub Form_Load()
'       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'       % Setup the System DSN, UserID, Password
'       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    dsn = "eprocess_svr_dsn"
    dbuser = "f_maint"
    dbpassword = "f_maint"

'       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'       % Create the Connection Object and open it
'       % with the supplied parameters
```

```
'       % System DSN, UserID, Password
'       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    Set connDB = CreateObject("ADODB.Connection")

    connDB.Open dsn, dbuser, dbpassword

'       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
'       % Build the SQL Statement and assign it
'       % to the variable SQLStatement. Execute
'       % the SQL statement
'       %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    SQLStatement = "SELECT * FROM f_sw.VWVQ262_Inbox"

    Set rs = connDB.Execute(SQLStatement)


   '          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   '          % Loop through Fields Names and print out the Field Names
   '          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

             For i = 0 To rs.Fields.Count - 1
             List2.AddItem (rs(i).Name)
             Next
   '          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   '          % Loop through rows, displaying each field
   '          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
             Do While Not rs.EOF
             For i = 0 To rs.Fields.Count - 1
             List1.AddItem (rs(i))
             Next
             List1.AddItem ("============================")
             rs.MoveNext
             Loop
   '          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   '          % Make sure to close the Result Set and the Connection object
   '          %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
             rs.Close
             connDB.Close

  End Sub
```

Back to top

Notices

Panagon documentation contains information proprietary to FileNET Corporation (FileNET). Due to continuing product development, product specifications and capabilities are subject to change without notice. You may not disclose or use any proprietary information or reproduce or transmit any part of this documentation in any form or by any means, electronic or mechanical, for any purpose, without written permission from FileNET. FileNET has made every effort to keep the information in the documentation current and accurate as of the date of publication or revision. However, FileNET does not guarantee or imply that the documentation is error-free or accurate with regard to any particular specification. In no event will FileNET be liable for direct, indirect, special incidental, or consequential damages resulting from any defect in the documentation, even if advised of the possibility of such damages. No FileNET agent, dealer, or employee is authorized to make any modification, extension, or addition to the above statements. FileNET may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Furnishing this document does not provide any license to these patents, trademarks, copyrights, or other intellectual property. FileNET is a registered trademark.

This product incorporates an Apache Software Foundation XML Parser. Apache assumes no liability for any claim that may arise regarding this incorporation. In addition, FileNET disclaims all warranties, both express and implied, arising from the use of the Apache XML Parser. http://www.apache.org/licenses/LICENSE.

Panagon Help systems incorporate Deva Search, a product of Deva Associates LLC. Deva Associates assumes no liability for any claim that may arise regarding this incorporation. Deva Search is a trademark or registered trademark of Deva Associates.

All other brands, products, and company names mentioned are trademarks of their respective owners.

eProcess Services Directory Structure

Much of the Panagon Web WorkFlo functionality relies on the Panagon Web Services architecture and directory structure. When they are installed together the Panagon Web Services and Panagon Web WorkFlo applications collectively are called eProcess Services.

During Panagon Web Services installation, a unique directory structure is created. The directory structure is extended during Panagon Web WorkFlo installation. This topic describes the directory structure created by Panagon Web Services and indicates the significant directory structure changes made during Panagon Web WorkFlo installation. Development decisions must account for the modified eProcess Services architecture and directory structure.

Note The directory structure shown here is relative to the **<drive>...\Program Files\FileNET\IDM\Web\IDMWS** directory on the Web WorkFlo server. Refer to Locate the Panagon Web Services Toolkit Help for more information on using Panagon Web Services components in eProcess Services development.

Directory	Contents
_ScriptLibrary	Contains files used for Microsoft Remote Scripting. These library files have been modified by FileNET.
\Application	Contains the modules used by eProcess Services, and the ASP files used to create the neighborhood, and eProcess application panes. Additionally, this directory contains the on-line help for both Panagon Web Services and Panagon Web WorkFlo.
\CSS	Contains Cascading Style Sheets (CSS) used eProcess Services. The style sheets control the text formatting and layout of the HTML content.
\Images	Contains many of the .GIF-formatted icon graphics used Panagon Web Services application, including the email notifications. Most of the .GIF files used by Web WorkFlo Services, like those used in Step Processors, are located in the **\Redist** directory, in the associated directory structure.

\Redist

Contains sub-folders which contain redistributeable components (like ASP, HTML, .JS, and .JAR files). These components provide the core functionality for eProcess Services.

The following directories were added during the Web WorkFlo installation:

- **\WF_Extras**: contains .ASP files and the **pw.jar** file. The .ASP files are used for attachment processing and Session-based token management. The **pw.jar** file is an archived file containing the class and image information used by Panagon WorkFlo applets and applications.
- **\WF_FileOpen**: contains .ASP files used for searching for and opening files stored in a Content Services library.
- **\WF_Html_Toolkit**: contains the .ASP files, .INC files, .JS files, and image files, in sub-directories, that constitute the HTML Processor Toolkit. Many of the files in this directory are shared by the default HTML processors. Refer to the HTML Processor Toolkit Overview topic for more information.
- **\WF_Launcher**: contains the .ASP source for the HTML and Java launch processors. The Java processor .ASP source is located in the **\Tabbed** directory; the ASP source is located in the **\HTML** directory. In addition, the HTML directory contains a sub-directory containing the image (.GIF) files used by the HTML processor. New Launch Step Processors must be added to a directory under this directory.
- **\WF_Logon**: contains .ASP and .INC files used to logon to a WorkFlo Server through a Panagon Web Services server. (The .INC file contains the Javascript functions that are called from the logon dialog box.
- **\WF_Manager**: contains the .ASP file that hosts the Personal WorkFlo Manager console application.
- **\WF_Milestone**: contains .ASP source for the milestone tracking components.
- **\WF_Step**: contains the .ASP source for the HTML and Java Step Processors. The Java processor .ASP source is located in the **\Tabbed** directory; the ASP source is located in the **\HTML** directory. New Step Processors must be added to a directory under this directory.
- **\WF_Tracker**: contains an .ASP file that acts as a container for the WorkFlo Tracker application.

The remaining directories were created during the Panagon Web Services installation:

- **\AddFolder**: contains an .ASP page used for folder creation operations.
- **\CheckoutList**: contains .ASP pages used for checkout pop lists.
- **\CopyMove**: contains an .ASP page used for copy and move operations.
- **\DocControls**: contains .ASP pages used for simple and compound document operations .
- **\DocWizard**: contains .ASP pages used for adding and checking in a document.
- **\Download**: extended by Web WorkFlo to include the Sun Java Runtime Environment Installation package. Contains specific executables, like the Panagon IDM Viewer, ready for downloading to client machines.
- **\ErrorManager**: contains .ASP pages used for reporting and handling errors.
- **\ListView**: contains .ASP pages used for displaying folder content.
- **\Logon**: contains .ASP pages used for logging on to Image Services (IS) or Content Services (CS) libraries.
- **\PopUpManager**: contains a Javascript file used to manage end-user message boxes and dialog boxes.
- **\PropertySheets**: contains .ASP pages used to determine and display object properties for both IS and CS libraries.
- **\Search**: contains .ASP pages used to support query operations.
- **\Toolbar**: contains .ASP pages used to display toolbars, which provides access to all operations.
- **\Treeview**: contains .ASP pages used to select and display the folder structure within both IS and CS libraries.
- **\Utility**: contains .ASP and .HTM pages which are used for a variety of operations.
- **\VersionList**: contains .ASP pages used to display version information for documents.

Most of the ASP files in the folders contain #include statements. You need to examine the beginning of each file to determine dependencies.

\WSAPI Contains .ASP files that implement the Panagon Web Services API objects and work with the Panagon IDM Object automation. Panagon Web WorkFlo extends some of the files in this directory.

\WSCAPI Contains the client scripts (.JS) files. Panagon Web WorkFlo extends some of the files in this directory.

Set up an ASP eProcess Services Development System

Panagon eProcess does not require that you use Visual InterDev to develop new or modify existing ASP files; however, Visual InterDev provides a robust ASP development environment. This topic describes how to use Microsoft Visual InterDev to create a new solution, add the necessary Panagon Web WorkFlo components, and begin developing for eProcess Services in an ASP environment.

Note This section covers developing for the ASP environment. For information on developing eProcess for the ASP.NET framework, see the "Developing for eProcess Open Client" section in the *Panagon Open Client Developer's Guide*.

One alternative to using an IDE or ASP development tool, like Visual InterDev, is to work directly in the **<drive>...\Program Files\FileNET\IDM\Web\IDMWS** directory structure on your eProcess Services web server. If you know HTML and ASP and you want to make only minor changes to specific files, working in the directory structure may be the simplest approach. (Since you will be working in an active web server directory structure, it is good practice to make a fallback copy of the directory before you start). If you are developing a larger application with multiple changes across many files, you should consider using a more sophisticated development tool, such as an IDE.

These procedures assume you have a WorkFlo Services server and a web server dedicated to development and configured with the following:

- Windows NT 4.0 or Windows 2000 Server
- Internet Information Server (IIS)
- FrontPage 98 or FrontPage 2000 Server Extensions (for remote development only)
- Panagon Web Services 3.x (with the appropriate service packs installed)

Note Refer to the *Web Server setup* section of the *Panagon Web Administrator's Help* and the *Configuration* section of the *IDM Web Administrator's Help* for installation requirements.

Additionally, these procedures assume you will be developing your ASP-based applications on a development system, rather than a production web server. In order to remotely develop your application, using Visual InterDev, you must install the FrontPage Extensions. If you choose develop your application directly on your production web server, you need not install the FrontPage Extensions. Refer to the Panagon eProcess Services architecture topic for more detailed information on the server configuration.

Note If you have not already installed IIS on your Windows NT Server, refer to the appropriate Microsoft documentation for the proper installation sequence. For information on installing Panagon Web Services, refer to the *IDM Web Administrator's Help* help.

Perform the following general steps to set up a Panagon Web WorkFlo server for development:

1. If you have not already done so, install the Panagon Web Services Web Application 3.x. (Refer to the *Web Server setup* section of the *Panagon Web Administrator's Help* for more information on installing Web Services and the Web Application.)

2. Using the **IDM Configure** application, add at least one library to your server. Refer to the *Configuration* section of the *IDM Web Administrator's Help* for more information on configuring your server to access Image Services (IS) and Content Services (CS) libraries. (If desired, eProcess can be configured without a requirement for Content Services library access. For information on configuring eProcess without Content Services, refer to the *Panagon WorkFlo Services Installation Handbook*.)
3. Install the Web WorkFlo Services software on your web server. (Refer to the *Panagon Web WorkFlo Installation Handbook* for installation instructions.)
4. On your WorkFlo Services server, configure the administration and user accounts to match your expected implementation. (Refer to the *Help for eProcess Administrators* for configuration instructions.)
5. Start the **eProcess Router** and **PPM** processes on your workflow server and web server (respectively).
6. Using a web browser (on a client system), verify that you can reach the Panagon Web Services home page by entering a command similar to the following address:

http://<web server name>/idmws/home.asp

where <web server name> is the host name of the server running IIS.

7. Click the **eProcess** bar in the left pane.
8. If the client system does not have the correct ActiveX components, the browser will be redirected to the client ActiveX component download page automatically. Depending on the client, you might need to reboot after installing the ActiveX components. Additionally, if the client system does not have the appropriate Java Runtime Environment installed (for details, see Panagon eProcess Services Architecture), the JRE will download from the server automatically the first time the client accesses the server.

You are ready to create a new solution in Visual InterDev.

Create a New Solution in Visual InterDev

These instructions assume you are developing on a system other than your eProcess Services web server. Refer to the appropriate Microsoft Developer's Network documentation for information on using Visual InterDev to develop applications in a distributed environment.

After you have installed an eProcess Services server successfully, you can create a new solution in Visual InterDev. If you have not already done so, you should install the FrontPage Extensions on your Internet Information Server (IIS) web server. With the FrontPage Extensions installed on the IIS server, you can develop your web application on a remote system and publish the updated application content directly to the root directory of your web application.

Create a new solution by performing the following steps:

1. If you have not already done so, open Visual InterDev.
2. From the menu bar, select **File**. From the pull-down list, select **New Project**.
3. With Visual InterDev Projects highlighted, select **New Web Project**. Supply a name and location for the project. For example, eProcess, and C:\My Documents\Visual Studio Projects\eProcess.
4. Click **Open**.
5. Select the WorkFlo web server you have already set up. Choose **Master Mode**. Click **Next**.
6. There might be a delay while Visual InterDev connects to the specified web server.
7. When prompted, enter a name for your new web application. For example, eProcess.

You can choose to connect to the existing web location and modify the out-of-the-box application directly; however, these instructions assume you are creating a new web application and will use only those parts of the eProcess Services components that are essential to make your application work with Image Services and Content Services libraries. Additionally, while you can choose to add the site to the root level of the web site, these procedures assume that you are developing your application in an isolated environment, which is separate from your production web site.

8. Click **Finish**.
9. When prompted, enter the user name and password for a user that belongs to the administrator group on the web server.

If your remote workstation and web server reside in different domains, you will have to supply the domain name as part of your user name. (For example, for a user in the Asgard domain, you would enter *Asgard*\<*user name*> in the **User** name field.)

10. Once all of the new files and folder are added to your new solution, select and highlight all directories and files below the bolded solution site address (in the Project Explorer pane).
11. From the **Edit** menu, select **Delete**. When prompted, select **Apply to all items** and click **Yes**.

All of the files and directories created for your new solutions should be deleted. You are now ready to add the eProcess Services components to your new solution.

Add eProcess Services Components

After creating a new solution and deleting the directories and files (created by Visual InterDev), you must add the eProcess Services components necessary to extend your application's existing functionality.

Add the necessary components by performing the following steps:

1. If you have not already done so, open Visual InterDev.
2. Locate and open your new solution.
3. Highlight the bolded solution site address in the **Project Explorer** pane (for example, your_webserver/your_webproject_name).
4. From the menu bar, click **Project** and select **Add Item**.
5. Click the **Existing** tab, and navigate to the mapped directory containing the Web Services toolkit files.

If you are developing your web application on a remote work station, you must map a network drive to the directory on the web server that contains the Web Services toolkit files. For example, if you installed the Panagon Web Services application to the default location, you would need to create a temporary network drive map to the **<drive>...\Program Files\FileNET\IDM\Web\IDMWS** directory on the web server.

6. From the **Files of type** pull-down list, select **All Files**.
7. Select and highlight all of the files in the list view box. Click **Open**.
8. Once the files from the root directory have been added to your new solution, click **Project** and select **Add Item** a second time.
9. Click the **Existing** tab, and navigate to the mapped directory containing the Web Services toolkit files.
10. Select and highlight all of the folders, including the **_ScriptLibrary** folder. (Refer to the eProcess Services Directory Structure topic for more information of the folders or components you need to add to your solution.)

If you want to use the Panagon Web Application with minor modifications, include the \Application folder in the list of folders to add to the solution.

11. Click Add Folder.

The selected folders, and their contents, are added to your new solution. The time necessary for the files to be copied to the local repository of the solution depends on the network and server speeds. Once the folders and files are added, the Project Explorer pane lists the files and folders copied into your new web solution. Your web project now has the necessary components for Web WorkFlo application development.

If you created a new application and new directory on your web server, you should check the web server directory permissions. You are now ready to create generic .ASP files for eProcess Services, modify HTML step and launch processors, or modify the .ASP file containers for the Java Step Processor and Launch Step Processors applets.

Check Web Server Directory Permissions

Calls to some Panagon Web Services URLs and COM components will fail unless you, or the server administrator, enable script execute permission for the directory containing your web application. Check (and if necessary set) the directory permissions by performing the steps on the appropriate platform.

Windows 2000

1. On your web server, point to **Start** > **Programs** > **Administrative Tools**.
2. Locate and click **Internet Services Manager**. The Internet Information Services (IIS) console displays.
3. In the IIS console, locate and open the directory name (node) associated with the solution you created (for example, eProcess). Right-click and choose **Properties**.
4. From the **Properties** dialog box, select the **Directory** tab, if it is not already on the top level.
5. Check that the **Read** option is selected and that the **Execute Permissions**, **Scripts and Executables** option is selected. If the options are already selected, click **Cancel** to exit the IIS console. Otherwise, select the options, and click OK.
6. Exit the Internet Services Manager console.

Windows NT

1. On your web server, point to **Start** > **Programs** > **Windows NT Option Pack** > **Microsoft Internet Information Server**.
2. Locate and click **Internet Service Manager**. The Microsoft Management Console (MMC) displays.
3. In MMC, locate the directory name (node) associated with the solution you created (for example, eProcess). Right-click and choose **Properties**.
4. From the **Properties** dialog box, select the **Directory** tab, if it is not already on the top level.
5. Check that the Read option and the Execute (including scripts) option are selected. If the options are already selected, click Cancel to exit MMC. Otherwise, select the options, and click **OK**.
6. Exit the Microsoft Management Console.

Create a Generic ASP File for eProcess Services

An Active Server Page (ASP) consists of HTML formatted text, or other content – like images, and server-side scripting (Javascript or VBScript). In many cases, the ASP includes client-side scripts. The HTML formatted content provides a framework for presenting dynamic content.

Custom applications can establish a session with the Panagon WorkFlo Services system through eProcess Services. Within the context of an ASP environment, you develop a web-based application that communicates to the IDM Objects through the Panagon Web Services JavaScript objects (for document functionality) and/or the JiGlue COM bridge (for workflow functionality). When working with the workflow functionality, you will use a scripting language (like VBScript or Javascript) in an ASP file to log on, create sessions, and retrieve information from a WorkFlo Services server through eProcess Services.

The script includes HTML tags, such as form tags, to interact with the browser user and script commands that contain logic for communication with the Panagon WorkFlo server. Since the scripting embedded in the ASP files runs on the server, only the HTML content (which might include client-side script) returns to the browser on the client system. The web server processes the server-side script and sends the HTML output to the client browser.

If you require user interaction, create a HTML form so the user can enter the necessary information. The ASP application can process the user input (for example, a request for queue names) and provide the appropriate information to the WorkFlo Services server.

Tip eProcess Services use the **global.asa** file to initialize the application state; however, you can use only one **global.asa** file for each ASP-based application on your web server.

Create web application (.ASP or .HTM) for eProcess Services by following these general steps:

1. If you have not already done so, open Visual InterDev.
2. Locate and open your new solution.
3. Highlight the bolded solution site address in the **Project Explorer** pane (for example, your_webserver/your_webproject_name).
4. From the menu bar, select **Project**. From the pull-down list, select **Add Item**.
5. With **Web Projects Files** highlighted, select the **ASP Page** icon. Alternately, if you want to add an HTML page, select the **HTML Page** icon.
6. In the **Name** field, enter the name for your new page. (The file extension is appended to the name automatically.)
7. Click **Open**.
8. Your new page displays in the **Project Explorer** pane, and the file contents display in development pane (in the source view).
9. Include the appropriate .JS and .ASP files.

Refer to the Referenced Panagon Web Services Files and the Referenced Panagon Web WorkFlo files topics for information on how and why to reference specific files. Refer to Locate the Panagon Web Services Toolkit Help for more information on using the Panagon Web Services components.

10. Include the appropriate HTML and scripting code.
11. Save your ASP page.

API Sample Overview

These API samples are a starting place for understanding and developing applications that use these APIs; other included samples (for example, the Java-based Step Processors and Launch Step Processors) cannot be correctly understood or modified without first understanding how to use the APIs as demonstrated in these samples. In general, these samples demonstrate the fundamental requirements of developing workflow processing applications (sometimes called work performers) that make efficient use of the APIs.

The API samples are documented extensively in this guide; however, this guide assumes that you are also looking at the sample source code for context. In addition to including topics explaining how the samples operate, the Run the unmodified samples topic details how to compile and run the MainSample sample, which is a simple workflow processing application. Many of the remaining topics include information on running samples from the command line.

The following table lists, in alphabetical order, the Java sample files and provides a brief description for each:

Sample file name	Description
LaunchSample.java	Demonstrates how to open and transfer a workflow definition file.
LockReportSample.java	Demonstrates how to log the work object number and user information for locked work objects.
Logger.java (helper class)	Provides reusable logging functions for the other sample files.
LogSample.java	Illustrates how to write the VWLog record information to a file.
LogViewer.java	Demonstrates how to create a log-viewing tool with "Select Log", "Create new log index", "Query Log", and "List Exposed Log Fields" options.
MainSample.java	Demonstrates how to develop a simple application in a modular fashion. This is a top-level application that calls main functions in the other sample files. These sample files create and launch a workflow definition, complete the steps, and write the history information to a log file.
MilestoneHelper.java (helper class)	Provides reusable milestone display-related methods that are referenced by some of the sample classes.

MilestoneSample.java	Reports milestone definitions in various workflows of a queue.
OperationsHelper.java (helper class)	Provides reusable methods for creating and reporting on operations on a queue.
OperationsSample.java	Creates operations on a queue and reports configuration information for all operations defined on a queue.
ProcessSample.java	Illustrates how to log process information.
QueueHelper.java (helper class)	Provides reusable queue-specific methods which are referenced by some of the sample classes.
QueueSample.java	Displays contents of workflow queues.
RosterHelper.java (helper class)	Provides reusable roster-specific methods which are referenced by some of the sample classes.
RosterSample.java	Demonstrates how to write the contents of a roster to a log file.
SessionHelper.java (helper class)	Provides reusable session-specific logon and logoff session methods that are referenced by some of the sample classes.
StepProcessorSample.java	Retrieves a step element from a queue, set the comments, displays the step information, and completes the step.
SysConfigSample.java	Illustrates how to create a queue, exposed field, and an index.
SystemStepHelper.java (helper class)	Provides reusable methods for building workflows with system steps.
SystemStepSample.java	Creates and launches a workflow with system steps.
UserInfoSample.java	Demonstrates how to retrieve and display user information.
WFDefinitionSample.java	Demonstrates how to create and save a workflow definition file.

WorkPerformerSample	Illustrates how to create a custom workflow processing application (also called a work performer) that polls a queue and dispatches work items.

LaunchSample

This sample demonstrates how to create a workflow definition object from the workflow definition file and transfer it to the database. Run the sample by entering a command similar to the following:

java LaunchSample username password <server name>:<port number>/<router instance name> [wfDefinition_filename | wfDefinition_filename output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The LaunchSample class contains two methods: the main(String args[]) method and the LaunchSample(VWSession vwSession, Logger logger, String wfDefFile) method, which is the constructor.

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default values for log output file and workflow definition file names are **LaunchSample.out** and **Sample.pep**, respectively. The main method constructs and passes vwSession and Logger objects to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session, the logger, the user name, and the name of the user definition file to the sample's class constructor.

LaunchSample(VWSession vwSession, Logger, String wfDefFile)

The sample constructor LaunchSample(VWSession vwSession, Logger, String wfDefFile) defines a workflow, transfers the definition to the system database, creates and comments the launch step, and dispatches the workflow as follows:

Create a workflow definition object from a file with VWWorkflowDefinition.readFromFile(). This presumes that local workflow definition (**Sample.pep**, for example) was created previously (for example, by the constructor for the sample class WFDefinitionSample).

```
wflDef = VWWorkflowDefinition.readFromFile(wfDefFile);
```

Transfer the definition with VWSession.transfer(...) and validate that the transfer was successful:

```
// Transfer the workflow definition.  The second parameter is a unique id.
```

```
// Transfer the workflow definition.  The second parameter is a unique id.
// We use a unique Content Services ID for this value,
// from the lib/docid /version.
```

```
VWTransferResult transferResult = m_vwSession.transfer(wflDef, "uniqueid", false, true);
   if (transferResult.success()){

                m_logger.log("The transfer was successful.");

        // ( less significant code . . . )

   }
   else
   {

         // Display the transfer errors.

      String[] errorArray = transferResult.getErrors();
      if (errorArray != null)
         m_logger.log("\tThe following transfer errors occurred: ", errorArray);
      else
         m_logger.log("\t\tError messages were not available.");
   }
```

Prepare to create a launch step by getting the version from the VWTransferResult object. Use the version string as the argument to instantiate a VWStepElement object ("launchStep") with the API method VWSession.createWorkflow(string). VWStepElement.setComment(string) to label this launch step as follows:

```
vwVersion = transferResult.getVersion();

// ( less significant code . . . )

launchStep = m_vwSession.createWorkflow(vwVersion);
launchStep.setComment("This is the sample workflow launch step comment");
```

Use the Logger object to log the launch step properties, which include the names and values of the parameters that were assigned to this step in a previous invocation of SysConfigSample.

```
m_logger.log("\nLaunch Step information:\n");
m_logger.log("\tWorkflow Name:  " + launchStep.getWorkflowName());
m_logger.log("\tSubject:  " + launchStep.getSubject());
m_logger.log("\tComment:  " + launchStep.getComment());
m_logger.log("\tStep Description:  " + launchStep.getStepDescription());

m_logger.log("\nParameters:\n");

paramNames = launchStep.getParameterNames();
if (paramNames == null)

        m_logger.log("\t\t no parameters!");

}
else { // ( less significant code . . . )
```

```
// Display the parameter names and their values.
for (int i = 0; i < paramNames.length; i++){
        if (paramNames[i] != null){
  // Retrieve the parameter value.
                value = launchStep.getParameterValue(paramNames[i]);
  // Write the information to the log.
                m_logger.log("\t" + paramNames[i] + " = ", value);
        }
}
```

Complete, or dispatch, the launch step with VWStepElement.doDispatch(). The dispatch saves the changes made to the step (comment properties, in this case) to the workflow database.

```
launchStep.doDispatch();
```

This sample, and all of the non-helper class samples, use the Logger and SessionHelper classes in the main method to log events and initialize the workflow session when it is run as a standalone application.

SessionHelper (helper class)

This is a helper class for other samples. This sample demonstrates how to implement reusable session-specific methods, logon and logoff, which can be accessed by other classes or applications. (The sample code creates a session from a router_URL and a user-supplied user name and password with appropriate access privileges.)

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

public SessionHelper(String user, String pw, String router, Logger)

The constructor code initializes variables for the user name, password, router_URL specification, and the logger object. Session logon and logoff are managed by the logon and logoff methods, respectively.

```
public SessionHelper(String user, String pw, String router, Logger){

m_userName = user;
m_password = pw;
m_routerPath = router;
m_logger = logger;

}
```

public VWSession logon()

This logon method invokes the VWSession.logon() method and checks for errors.

```
try
{

// The following code initializes a session object.

m_vwSession = new VWSession(m_userName, m_password, m_routerPath);
}
catch (Exception ex)
{
if (m_logger != null)
  m_logger.log(ex);
else
  ex.printStackTrace();
}
return m_vwSession;
```

public void logoff()

The logoff method invokes the VWSession.logoff() method and checks for errors. It can either log exceptions or print a stack trace, as shown below:

```
try {

// Logoff the session.

if (m_vwSession != null)
    m_vwSession.logoff();
}
catch (Exception ex){
  if (m_logger != null)
      m_logger.log(ex);
  else
      ex.printStackTrace();
}
finally {
  m_vwSession = null;
}
```

Logger (helper class)

This is a helper class to implement logging functions for other sample classes.

Note The sample code assumes you created a session and supplied a user name and password with appropriate access privileges.

Methods

Logger(String outputFile)

The calling program passes one argument when it invokes the constructor for the logger helper class sample. The parameter is a string variable (outputFile) that names the log file to open. The constructor creates an instance of a PrintWriter object ("m_fOut") and ends by displaying the name of the output file with the following code:

```
m_fOut = new PrintWriter(new BufferedWriter(new FileWriter(new File(outputFile))), true);
System.out.println("Writing messages to file: " + outputFile);
```

displayStepElementInfo(VWStepElement vwStepElement) and displayWorkObjectInfo(VWWorkObject vwWorkObject)

The logger object displays descriptive information for stepElement or WorkObject objects with Logger.displayStepElementInfo(object) or Logger.displayWorkElementInfo(object), respectively, using the object to be described as an argument. In each method the logger assembles display lines with the argument's (stepElement or workObject) retrieval properties, as in the following Logger.displayStepElementInfo() code:

```
String svalue = vwStepElement.getWorkObjectName();
log("\t\t\t"+ "WorkObjectName" + "=" + svalue);

svalue = vwStepElement.getTag();
log("\t\t\t"+ "Tag" + "=" + svalue);
```

log(Exception ex)

Prints a stack trace with ex.printStackTrace().

log(String text, Object arg1)

If argument "Object arg1" is NULL, the program invokes log(String text), which is described in the next method description, below. If "Object arg1" is a string, the method concatenates the two string arguments and invokes log(String text).

If argument "Object arg1" is an object or object array argument (usually an array of numerical values),

the method builds a string from the object using StringBuffer() methods as follows:

Make an array for the argument parameter and create a null Stringbuffer.

```
Object[] args = (Object[])arg1;
StringBuffer buffer = new StringBuffer();
```

Append an opening format string.

```
buffer.append("{");
```

Append each argument to the StringBuffer() object.

```
for (int i = 0; i < args.length; i++) {
```

Append a format string.

```
        if (i > 0) buffer.append(",");
                buffer.append(args[i]);
}
```

Append a closing format string.

```
buffer.append("}");
```

Apply StringBuffer.toString() to convert elements to strings and call log(String text, Object arg1) again. Note the ultimate result of this will be to concatenate the two strings and invoke log(String text).

```
log(text, buffer.toString());
```

log(String text) and logAndDisplay(String text)

Logger.log(string svalue) logs whatever the "svalue" string contains. "svalue" is often a string form of a data element that belongs to an object. You may wish to concatenate explanatory and formatting information in the argument; for example:

```
Logger.log("\t\t\t"+ "Tag" + "=" + svalue);
```

For each line passed to Logger.log(string text), the method invokes an instance of the printWriter object (m_fOut).

```
 if   (m_fOut != null)m_fOut.println(text);
 else System.out.println(text);
```

logAndDisplay(String text) uses the same code and adds a standard system display method as follows:

```
System.out.println(text);
```

WFDefinitionSample

This sample demonstrates how to create and save a workflow definition in a file. Run the sample by entering a command similar to the following:

java WFDefinitionSample username password <server name>:<port number>/<router instance name> [wfDefinition_filename | wfDefinition_filename output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default values for log output file and workflow definition file names are **WFDefinitionSample.out** and **Sample.pep**, respectively. The main method sets local variables to point to the VWSession and Logger object arguments and passes them to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session, the logger, the user name, and the name of the user definition file to the constructor for the class.

WFDefinitionSample(VWSession vwSession, Logger, String userName, String wfDefFile)

The constructor WFDefinitionSample(VWSession, Logger, String, String) performs common exception handling and demonstrates a variety of workflow definition functions. It creates the default workflow definition, sets some the workflow definition general properties, and creates some fields. It sets some properties for the main map, sets the description for the main map and the launch step, and creates some steps in the main map. Finally, it writes the validated workflow definition to a file. The code is organized as follows:

Create the default workflow definition:

workflowDef = new VWWorkflowDefinition();

Set subject, description, and name properties in the workflow definition:

```
workflowDef.setSubject("\"This is the sample workflow definition subject\"");
workflowDef.setDescription("This is the sample workflow definition description");
workflowDef.setName("Sample Workflow");
```

Create an integer and a string field in the workflow definition:

workflowDef.createFieldUsingString("Field1_Integer", "99", VWFieldType.FIELD_TYPE_INT, false);

```
workflowDef.createFieldUsingString("Field2_String", "{\"\"}", VWFieldType.FIELD_TYPE_STRING, true);
```

Set some properties for the map:

```
mapDef = workflowDef.getMainMap();
mapDef.setDescription("This is the sample workflow map");
```

Get the launch step and set its description:

```
currentStepDef = mapDef.getStartStep();
currentStepDef.setDescription("This is the description for the launch step.");
precedingStepDef = currentStepDef;
```

Iteratively create three steps in the "Workflow" (main) map with the local addStep method.

```
      for (int i = 0; i < 3; i++)
{
   currentStepDef = addStep(mapDef, "Step" + i, userName);
   if (currentStepDef != null){

            // create a route

      precedingStepDef.createRoute(currentStepDef.getStepId());

            // reset the preceding step

      precedingStepDef = currentStepDef;
   }
}
```

Write the validated workflow definition to a file:

```
if (validate(workflowDef, vwSession)){
    logger.log("Writing workflow definition to file: " + wfDefFile);
    workflowDef.writeToFile(wfDefFile);
}
```

Additional code in this method performs common user notification, cleanup, and exception handling.

private VWStepDefinition addStep(VWMapDefinition mapDef, String stepName, String userName)

Creates and initializes a workflow step. The code is organized as follows:

Create a destination step for a new route:

```
  newStepDef = mapDef.createStep(stepName);
  nStepId = newStepDef.getStepId();
```

Create an integer parameter and a string parameter. The last argument for the second VWStepDefinition.createParameter(...) is set to true because the programmer expects the value expression for this parameter to evaluate to an array.

```
  newStepDef.createParameter( "Field1_Integer", VWModeType.MODE_TYPE_IN, "99",
VWFieldType.FIELD_TYPE_INT, false );
  newStepDef.createParameter( "Field2_String", VWModeType.MODE_TYPE_OUT, "Field2_String",
VWFieldType.FIELD_TYPE_STRING, true );
```

Create a named participant and add it to the step

```
  participants = new VWParticipant[1];
  participants[0] = new VWParticipant();
  participants[0].setParticipantName("\"" + userName + "\"");

  newStepDef.setParticipants(participants);
```

Set the queue name, deadline, and reminder times.

The deadline value represents the number of minutes between receipt of the step by an application and expiration of the step's deadline timer at runtime. The reminder value represents the number of minutes prior to expiration of the deadline timer at which the reminder timer expires at runtime:

```
  newStepDef.setQueueName("Inbox");
  newStepDef.setDeadline(1000);
  newStepDef.setReminder(500);
```

Set the step description and design display location. The location coordinates in this example are arbitrary.

```
  newStepDef.setDescription("This is the description for step" + nStepId + ".");
 newStepDef.setLocation(new java.awt.Point(nStepId * 100, 150));
```

Write out the successful step-creation status.

```
  m_logger.log("Creation and initialization of step: " + nStepId + " is complete.");

}
```

Additional code in this method handles common cleanup, exception handling, and the return statement for the new VWStepDefinition object.

private boolean validate(VWWorkflowDefinition workflowDef, VWSession)

This method provides common exception and message handling when invoking the VWStepDefinition.validate(...) method. The "false" value of the second parameter of workflowDef.validate() will cause the system to assume that the caller has already set any join step IDs that may exist. Since there are no join step IDs in the example, this enhances performance.

```
  VWValidationError[] validationErrors = null;
```

```
validationErrors = workflowDef.validate(vwSession, false);
```

SysConfigSample

This sample demonstrates how to create a queue, exposed fields, and an index. Run the sample by entering a command similar to the following:

java SysConfigSample username password <server name>:<port number>/<router instance name> [output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The SysConfigSample class contains two methods: the main(String args[]) method and the SysConfigSample(VWSession vwSession, Logger logger) method, which is the constructor method.

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default value for the log output file is **SysConfigSample.out**. The main method constructs and passes vwSession and Loggerobjects to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session and the logger to the constructor method.

SysConfigSample(VWSession vwSession, Logger logger)

The constructor SysConfigSample(VWSession, Logger) performs common exception handling and demonstrates a variety of system configuration methods. It displays various SysConfig parameters, creates a test queue, adds one exposed field (integer), and adds one index. The code that performs these tasks is organized as follows:

Fetch the system configuration object, sysConfig, using the session object:

```
sysConfig = vwSession.fetchSystemConfiguration();
```

Use the system configuration object to get three of the system configuration parameters. Display them with the logger.log method:

```
logger.log("Logging state = " + sysConfig.getLoggingState());
logger.log("Max DB operations = " + sysConfig.getMaxDBOperations());
logger.log("Max instructions = " + sysConfig.getMaxInstructions());
```

Create a user-centric test queue:

```
// Create a test queue; add one exposed field (an integer) and one index.
```

```
queueDef = sysConfig.createQueueDefinition("test", VWQueue.QUEUE_TYPE_USER_CENTRIC);
if (queueDef != null){
```

Add an exposed field ("field1"):

```
VWExposedFieldDefinition exposedFieldDef = queueDef.createFieldDefinition("field1", 1, 0);
```

Create the index:

```
String[] fNames = {new String("field1")};
VWIndexDefinition id = queueDef.createIndexDefinition("index1", fNames);
```

Inform the user in the log:

```
logger.log("The queue 'test' was created with one exposed field ('field1') and one index ('index1').");
```

Commit the system configuration changes:

```
String[] errors = sysConfig.commit();
```

Finally, the following code logs what happened:

```
if (errors != null){
   logger.log("Errors: ", errors);
} else
   logger.log("All changes have been committed.");
```

Additional code for SysconfigSample performs common cleanup and exception handling.

LockReportSample

This stand-alone sample demonstrates how to log the work object number and user information for locked work objects. Run the sample by entering a command similar to the following:

java LockReportSample username password <server name>:<port number>/<router instance name> [output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The LockReportSample class contains two methods: the main(String args[]) method and the LockReportSample(VWSession vwSession, Logger logger) method, which is the constructor.

main(String args[])

The main method uses common techniques for validating and defaulting argument values. If the user does not supply an output filename, the main method supplies **LockReportSample.out**. The method sets local variables to point to the VWSession and Logger object arguments and passes them to the sample constructor. This method handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. The main method provides workflow logging functions by creating an instance of the sample Logger class. The LockReportSample main method passes the session and the logger instances to the LockReportSample constructor.

The following code from the LockReportSample main method enables it to operate as a stand-alone program. It validates the arguments and supplies a default value for the [output_filename] string, as needed.

```
if (args.length < 3 || (args.length > 0 && args[0].compareTo("?") == 0)){

        System.out.println("Usage:  LockReportSample username password
        router_URL [output_filename]");
        System.exit(1);

}

 // The file name (for output) is optional.

if(args.length > 3) fileName = args[3];
else

// Supply a default output_filename.

        fileName = new String("LockReportSample.out");
```

Instantiate Logger and session objects. The sample SessionHelper class instantiates the session object:

```
logger = new Logger(fileName);

// Create the session instance and log on.

sessionHelper = new SessionHelper(args[0], args[1], args[2], logger);
```

Log onto the session with sessionHelper.logon() and construct an instance of the LockReportSample class.

```
vwSession = sessionHelper.logon();
if (vwSession != null) {

        // Create the sample class.

   sampleClass = new LockReportSample(vwSession, logger);
}
```

When the class terminates, the main method terminates the session with sessionHelper.logoff(). Additional code in the main method manages common cleanup and error handling.

LockReportSample(VWSession vwSession, Logger logger)

The code for the LockReportSample constructor is organized as follows:

Output a header for the lock report withlogger.logAndDisplay(String):

```
logger.logAndDisplay("\n~ Begin writing lock report.");
```

Create a QueueHelper object:

```
queueHelper = new QueueHelper(vwSession, logger);
```

Log the lock status of each work object in each queue.

```
queueHelper.displayQueueLockStatus();
```

QueueHelper (helper class)

This is a helper class with reusable methods for other samples or applications, particularly the QueueSample. The sample demonstrates how to get queue information and report on various queue-related objects.

Note The sample code assumes you created a session and supplied a user name and password with appropriate access privileges. The QueueHelper(VWSession, Logger) constructor sets local variables to point to its SessionHelper and Logger objects, which are initialized by the main method. The QueueHelper.main() code is as follows:

```
public QueueHelper(VWSession vwSession, Logger){
m_vwSession = vwSession;
m_logger = logger;
}
```

Methods

The following sections describe the public QueueHelper class methods.

displayQueueContents(VWQueue vwQueue)

Displays the contents of a Queue: properties of queue elements, step elements, and work objects. This method retrieves the queue information using methods displayQueueElements(...), displayStepElements(...), and displayWorkObjects(...), which are described below.

displayQueueElementInfo(VWQueueElement vwQueueElement)

Displays the properties of a single queue element with the following steps:

Validate the input VWQueueElement as follows:

```
if (vwQueueElement == null){
m_logger.log("\t The queue element is null!");
return;}
```

Set a string array (fieldNames) to the names of the system fields as follows:

```
fieldNames = vwQueueElement.getSystemDefinedFieldNames();
```

Validate that there were system fields:

```
if (fieldNames == null){...}
```

Log the name and value of each system field:

```
} else {
```

```
        m_logger.log("\t\tSystem Defined Fields:");
        for (int i = 0; i < fieldNames.length; i++)
        {

                if (fieldNames[i] != null){

                value = vwQueueElement.getFieldValue(fieldNames[i]);

// Display the field name and its value.

                m_logger.log("\t\t\t" + fieldNames[i] + "=" + value);

    }
  }
}
```

User-defined fields are displayed in the same way that is shown above, but the field names array is set with a different method as follows:

fieldNames = vwQueueElement.getUserDefinedFieldNames();

Properties of the queue element are logged individually. For example, the queue element work object number, the work object name, and the work object tag are logged as follows:

```
String bvalue = vwQueueElement.getWorkObjectNumber();
m_logger.log("\t\t\t"+ "WorkObjectNumber" + "=" + bvalue);

String svalue = vwQueueElement.getWorkObjectName();
m_logger.log("\t\t\t"+ "WorkObjectName" + "=" + svalue);

svalue = vwQueueElement.getTag();
m_logger.log("\t\t\t"+ "Tag" + "=" + svalue);
```

Other queue element methods are used to get additional property values, and the results are logged (as shown above). This method uses the following queue element methods to obtain property values: getWorkClassName(), getQueueName(), getOperationName(), getLockedStatus(), and getLockedMachine().

displayQueueElements(VWQueue vwQueue)

Queries the series of all queue elements. displayQueueElements(...) performs the following steps:

The first step to query a series of all queue elements is to create a buffer for the items retrieved in each server fetch transaction with the following code:

```
// Set the maximum number of items to be retrieved in each server
// fetch transaction. In this case, setting the value to 25 will require
// less memory for each fetch than the default setting (50).

vwQueue.setBufferSize(25);
```

Construct a queue query object and query for all elements.

```
qQuery = vwQueue.createQuery(null, null, null, 0, null, null,
VWFetchType.FETCH_TYPE_QUEUE_ELEMENT);
```

Fetch the first queue element using the VWQueueQuery object. Determine is any queue elements exist.

```
vwQueueElement = (VWQueueElement)qQuery.next();
if (vwQueueElement == null){
m_logger.log("\t Queue elements: none");
```

If there are queue elements, the method uses local method displayQueueElementInfo() to display the queue element information as follows:

```
} else {
do {

  // display the queue element information

  displayQueueElementInfo(vwQueueElement);
}
```

Assign a new queue element with the next() method and continue the loop, if there is another element:

```
while ((vwQueueElement = (VWQueueElement)qQuery.next()) != null);
}
```

displayQueueLockStatus() (below) performs a similar query of all queue elements, but it differs by controlling the loop with the qQuery.next() method.

displayQueueLockStatus()

Displays lock status information for all workflow queue elements. The method logs queue names with the processing steps described below.

Put the queue names into string array queueNames, using QueueHelper.getQueueNames(...) and QueueHelper.getQueue(...) (described separately below), as follows:

```
queueNames = getQueueNames(false);
if (queueNames != null){

        m_logger.log("Found " + queueNames.length + " queues (NOT including system
        queues):");
```

Display each queue name as follows:

```
        for (int i = 0; i < queueNames.length; i++){


 // Get the queue object

                vwQueue = getQueue(queueNames[i]);

 // Display queue name
```

```
            m_logger.log("\nQueue:  " + vwQueue.toString());

//  Note that the queue name could also be displayed with:
//  m_logger.log("\nQueue:  " + queueNames[i]);
```

For each queue, set the maximum number of items to be retrieved in each server fetch (query) transaction. In this case, setting the value to 25 will require less memory for each fetch than the default setting (50).

```
            vwQueue.setBufferSize(25);
```

Construct a queue query object and query for all locked elements. Note that the input flag limits the returned object list to locked objects only. The query flag is read and lock, the filter expression is "F_Locked = 1", and the fetch type is the Queue Element.

```
            qQuery = vwQueue.createQuery(null, null, null, 1, "F_Locked = 1", null, 3);
```

Note that this is equivalent to:

```
            qQuery = vwQueue.createQuery(null, null, null, QUERY_READ_LOCKED,
            "F_Locked = 1", null, FETCH_TYPE_QUEUE_ELEMENT);
```

Query a series of queue elements for work object number with the VWQueueQuery.next(), VWQueueQuery.hasnext(), queueElement.getWorkObjectNumber(), and VWQueueElement.getLockedUser() methods and log the result:

```
            if (qQuery.hasNext()){

// Display column headers for list of locked workobjects.

                  m_logger.log("\t WorkObject Number\t\t\tLocked User");
                  do {

// Fetch the next queue element using the VWQueueQuery object.

                        queueElement = (VWQueueElement)qQuery.next();

// Display the workobject number of each locked workobject.

                        m_logger.log("\t " +
                        queueElement.getWorkObjectNumber() + "\t" +
                        queueElement.getLockedUser());

                  }while (qQuery.hasNext());

            } . . .

   }

}
```

displayQueueNames()

Logs all queue names with the following steps:

Get a string array of queue names:

queueNames = getQueueNames(true);

Invoke the logger.log method to log each one:

```
for (int i = 0; i < queueNames.length; i++){
m_logger.log("\t\t" + queueNames[i]);
}
```

displayStepElements(VWQueue vwQueue), displayStepElements(VWQueue vwQueue), and displayWorkObjects(VWQueue vwQueue)

These methods display property information for a group of queue, step, or work elements (work objects). The processing steps these methods are similar to each other. All the methods begin by setting up an iterative query of the element group as follows:

Validate the VWQueue object:

```
if (vwQueue == null){
m_logger.log("The queue object is null!");
return;
}
```

Set the maximum number of items to be retrieved in each server fetch transaction. Setting the value to 25 will require less memory for each fetch than the default setting (50).

vwQueue.setBufferSize(25);

Construct a query object for the type of object group being processed. The query object is instantiated in accordance appropriately with the last input parameter (fetchType) of the VWQueue.createQuery method as shown below:

for step elements

```
qQuery = vwQueue.createQuery(null, null, null, 0, null, null,
VWFetchType.FETCH_TYPE_STEP_ELEMENT);
```

for work elements

```
qQuery = vwQueue.createQuery(null, null, null, 0, null, null,
VWFetchType.FETCH_TYPE_WORKOBJECT);
```

for queue elements

```
qQuery = vwQueue.createQuery(null, null, null, 0, null, null,
```

```
VWFetchType.FETCH_TYPE_QUEUE_ELEMENT);
```

Fetch the first element using the VWQueueQuery object:

```
vwElement = (VWStepElement)qQuery.next();
```

Check to see if there are any queue elements:

```
if (vwElement == null){
m_logger.log("\t Number of elements: none");
}
else
{
```

Iteratively examine each element and display appropriate properties for the element type:

```
   do {

      // Display the queue element information with one of the following:
      // displayQueueElementInfo(vwElement); // for queue elements, or:
      // displayStepElementInfo(vwElement);  // for step elements, or:
      //displayWorkObjectInfo(vwElement);  // for work objects

   }

// Advance to the next element with one of the following while loop footers:
//  while ((vwQueueElement = (VWQueueElement)qQuery.next()) != null); // or
//  while ((vwQueueElement = (VWStepElement)qQuery.next()) != null); // or
//  while ((vwQueueElement = (VWWorkObject)qQuery.next()) != null);

}
```

VWQueue getQueue(String queueName)

Return the queue object for the queueName requested, after Initializing VWQueue object (vwQueue) and setting it as follows:

```
vwQueue = m_vwSession.getQueue(queueName);

//( Error handling code . . . )

return vwQueue;
```

String[] getQueueNames(boolean bIncludeSystem)

Returns an array of queue names and shows how to use session object flags to control what queue names are fetched, with the following processing steps:

Initialize the fetch flags variable (nflags) and a string array (queueNames)

```
int nFlags = VWSession.QUEUE_PROCESS | VWSession.QUEUE_USER_CENTRIC |
```

```
VWSession.QUEUE_IGNORE_SECURITY;
String[] queueNames = null;
 Use the passed in flag (bIncludeSystem) to optionally include system queues:
if (bIncludeSystem) nFlags |= VWSession.QUEUE_SYSTEM;
Retrieve the list of available queues with VWSession.fetchQueueNames(int)
queueNames = m_vwSession.fetchQueueNames(nFlags);
// ( error handling code . . . )
return queueNames; // (a string array)
```

LogSample

This sample demonstrates how to write the process default event log record information to a file. Run the sample by entering a command similar to the following:

java LogSample username password <server name>:<port number>/<router instance name> [output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The LogSample class contains two methods: the main method main(String args[]) and the constructor method LogSample(VWSession vwSession, Logger logger).

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default value for the log output file is "LogSample.out". The main method constructs and passes vwSession and Logger objects to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session and the logger to the sample's class constructor.

Logsample(VWSession vwSession Logger logger)

The constructor for the LogSample class notifies the user of its work and creates a log query object for the default event log. It validates the object, its elements, and the element fields and displays element and field information. The LogSample constructor performs this as follows:

Notify the user.

```
logger.logAndDisplay("\n~ Starting LogSample execution.");
```

Get the log object for the DefaultEventLog and log its name.

```
vwLog = vwSession.fetchEventLog("DefaultEventLog");
logger.log("Log: " + vwLog.toString());
```

Set the maximum number of elements to 25, which is half the default value of 50. A value of 25 will require less memory for each fetch.

```
vwLog.setBufferSize(25);
```

Construct a log query object and query for all elements. Create a VWLog class query with the VWLog.startQuery() method. Note that this contrasts with queue and roster query objects, which are

constructed by createQuery() methods.

```
logQuery = vwLog.startQuery(null, null, null, 0, null, null);
logElement = logQuery.next();
```

Check to see if there are any log elements.

```
if (logElement == null){
logger.log("\t Log elements: none");
}
else { . . .
```

Iterate through the log elements. Log each element to the output file.

Put the element's field names into a string array with the VWLogElement.getFieldnames() method, and output a message if no fields are found.

```
do {
logger.log("\t Log element:");

   // Display the fields.

        fieldNames = logElement.getFieldNames();

        if (fieldNames == null){

                logger.log("\t\t No fields!");

        } else {

                logger.log("\t\t Fields:");
```

Retrieve and display each field name and its value. VWLogElement.getFieldValue(string) returns an object that is the corresponding value for the fieldnames.

```
    for (int i = 0; i < fieldNames.length; i++){

                if (fieldNames[i] != null){

  // Retrieve the field value object.

                        value = logElement.getFieldValue(fieldNames[i]);
```

Display a value string for the field value object by invoking logger.log(String, Object).

```
// Display the field name and value(s).  Note that this log method takes an object for its second parameter.

                        logger.log("\t\t\t" + fieldNames[i] + " = ", value);

                }

        }
```

```
  }
while ((logElement = logQuery.next()) != null);
}
} // . . .
```

Additional code handles messages, exceptions, and cleanup.

LogViewer

This sample demonstrates how to create a log viewing tool that uses Swing components to render the user interface. Run the sample by entering the following command line:

java LogViewer

The main method invokes the constructor VWSample3A(), which sets up a dialog box that is the user interface to the log viewing tool. The dialog contains a system menu with "Select system/router", "logon", and "logoff" items, and it contains a log menu with "Select Log", "Create new log index", "Query Log", and "List Exposed Log Fields" items. The program responds to each of these menu selections according to the code in the actionPerformed method. Each program response is described in its own section below.

Select system/router

Selecting this menu choice allows the user to choose the system and router to which to connect. The input must be in the following form:

<server_name>:<port_number>/<router_instance_name>

The above format is identical to that of the router_URL parameter in the API VWSession() constructor or VWSession.logon. For a detailed explanation of the command, see the Run the sample application section of the Run the Unmodified Samples topic.

Logon

A Swing component collects the username and password. The program passes this information, along with the router_URL from the router selection (Select system/router, above) to the API VWSession constructor, as follows:

s = new VWSession(uname, pword, routerName);

Logoff

Invoke the VWSession.logoff method to free system resources as follows:

s.logoff();

Object "s" represents the instance of the session that was created at the time of the logon.

Select Log

The program creates an array of strings containing the log names with the API VWSession.fetchEventLogNames() method as follows:

String[] elogNames = s.fetchEventLogNames();

The program opens a dialog box that prompts the user to select a log by name, and when a name is selected, it creates an instance of the API VWLog class with that name and notifies the user with the following code:

```
if (logName != null){
 try
 {
    vw_Log = s.fetchEventLog(logName);
    isLogSelected = true;
    JOptionPane.showInternalMessageDialog(dpane, "Opened " + logName);
 }
 catch (Exception e)
 {
    e.printStackTrace();
    JOptionPane.showInternalMessageDialog(dpane, "Unable to open " + logName);
    isLogSelected = false;
 }
}
```

Object "s" above represents the instance of the session that was created at the time of the logon.

Query Log

When the user makes this menu selection, the program first creates a VWLogDefinition object with the VWLog.fetchLogDefinition() method. The VWLogDefinition.getIndexes() method creates an array of VWIndexDefinition objects as follows:

```
VWLogDefinition vw_LogDef = vw_Log.fetchLogDefinition();
VWIndexDefinition[] vw_Indexes = vw_LogDef.getIndexes();
```

Using the VWIndexDefinition object array, the program creates a dialog box from which the user may select an index for an index query. Store the select index in a VWIndexDefinition object name vw_Idx:

```
VWIndexDefinition vw_Idx = vw_Indexes[xix];
```

The next dialog the program displays asks the use to select a query method cancel. Cancellation results in the program returning from the LogViewer method; otherwise, the user selects either "Basic Query" or "Index-Based Query", and the program creates a log query object. Log query object vwq is initialized as follows:

```
VWLogQuery vwq = null;
```

If the user selects Index-based query, the logViewer method creates another dialog box that enables the user to select maximum and minimum values for each component field of the selected index. The first step is to load a string array with a sorted list of index field names, using the VWIndexDefinition.getFieldNames() method as follows:

```
String[] idxFieldNames = vw_Idx.getFieldNames();
Arrays.sort(idxFieldNames);
```

The code below creates an array of VWExposedFieldDefinition objects containing the exposed log definition fields, using the VWLogDefinition.getFields() method. The next line initializes an array of VWExposedFieldDefinition objects, fieldDefs, with the same number of elements.

```
VWExposedFieldDefinition[] vw_EFDef = vw_LogDef.getFields();
VWExposedFieldDefinition[] fieldDefs = new VWExposedFieldDefinition[idxFieldNames.length];
```

The objects for the exposed field definitions that are part of the selected index are loaded into the empty array fieldDefs[] with the same index as the index field names as follows:

```
for (int i=0;i<vw_EFDef.length;i++){
  int x = Arrays.binarySearch(idxFieldNames, vw_EFDef[i].getName());
  if (x >= 0){
     fieldDefs[x] = vw_EFDef[i];
  }
}
```

A dialog box displays field definitions and their types with text box pairs for the user to fill in minimum and maximum values. The program copies these minimum and maximum values into two object arrays, one for minimum values and the other for maximum values, respectively. These arrays, min and max, are arguments for the VWLOG.startQuery() method to create a log query object, as shown below:

```
vwq = vw_Log.startQuery(vw_Idx.getName(), min, max, 0, null, null);
```

If the Basic Query method is chosen, the program immediately creates the log query object. The min and max object arrays are replaced by NULLs, as follows:

```
vwq = vw_Log.startQuery(vw_Idx.getName(), null, null, 0, null, null);
```

The program employs a logElement object to load information from the log query object into ArrayList objects as follows:

```
VWLogElement le = null;                 // Initialize data targets:
ArrayList eventTypes = new ArrayList();
ArrayList timeStamps = new ArrayList();
ArrayList seqNumber = new ArrayList();  // Load data targets:

while (vwq.hasNext()){
  le = vwq.next();
  eventTypes.add(VWLoggingOptionType.getLocalizedString(le.getEventType()));
  timeStamps.add(le.getTimeStamp());
  seqNumber.add(String.valueOf(le.getSequenceNumber()));
}
```

In the code above, the VWLogElement methods, getEventType(), getTimeStamp(), and getSequenceNumber() extract events, time stamps, and sequence numbers from an iterated series of log elements that are derived from the log query object.

The three types of information for each log element are loaded into a two-dimensional array as follows:

```
Object[] e1 = eventTypes.toArray();
Object[] e2 = timeStamps.toArray();
Object[] e3 = seqNumber.toArray();

Object[][] data = new Object[e1.length][3];

for (int i=0;i<e1.length;i++){
  data[i][0] = e1[i];
  data[i][1] = e3[i];
  data[i][2] = e2[i];
}
```

Finally, the program uses this data array as input to Swing components that display event type, sequence number, and time stamp in a table, one row for each log element.

Create new log index

Indexes on exposed log fields serve to speed up queries, just as indexes on databases do. When the "Create new log index" option is selected, the program uses the API VWSystemConfiguration and VWLogDefinition classes to handle index information. Log index creation begins by creating a VWSystemConfiguration object with the VWSession.fetchSystemConfiguration method and a VWLogDefinition object with the VWLog.fetchLogDefinition() method:

```
VWSystemConfiguration sysConfig = s.fetchSystemConfiguration();
VWLogDefinition vw_LogDef = vw_Log.fetchLogDefinition();
```

The VWLogDefinition object method, vw_LogDef.getFields(), loads an exposed field definition object, vw_EFDef, whose getname() method subsequently loads a string array, EFDefNames[]:

```
VWExposedFieldDefinition[] vw_EFDef = vw_LogDef.getFields();

for (int i=0;i<vw_EFDef.length;i++){
  EFDefNames[i] = vw_EFDef[i].getName();
}
```

A Swing component dialog prompts the user to choose indexes and specify a name. The components output an array of indexes, idx[], to the Exposed Field Definition Names array. The program loads string array "fields" with the chosen index names as follows:

```
String[] fields = new String[idx.length];
for (int i=0;i<idx.length;i++){
  fields[i] = EFDefNames[idx[i]];
}
```

The program saves the user specified name in a string variable, "name". VWLogDefinition.createIndexDefinition(..., ...) uses this name and the fields string array created above to create an index for the log definition as follows:

```
vw_LogDef.createIndexDefinition(name, fields);
```

Finally, the program updates the system configuration with the new log definition. Note that the configuration must be committed with the VWSystemConfiguration.commit method:

```
sysConfig.updateLogDefinition(vw_LogDef);
String[] err = sysConfig.commit();
```

The program can use the returned string array for common error handling.

List Exposed Log Fields

The code for this selection shows how to create a list of all exposed fields in the currently selected log. The program creates three objects: a VWLogDefinition object, a VWExposedFieldDefinition object array, and a String array of exposed field names. The VWLog.fetchLogDefinition() method loads the log definition object, and the VWLogDefinition.getFields() method loads an array of exposed fields as follows:

```
VWLogDefinition vw_LogDef = vw_Log.fetchLogDefinition();
VWExposedFieldDefinition[] vw_EFDef = vw_LogDef.getFields();
```

The exposed field's names are copied into a string array with the following code:

```
String[] EFDefNames = new String[vw_EFDef.length];

for (int i=0;i<vw_EFDef.length;i++){
  EFDefNames[i] = vw_EFDef[i].getName();
}
```

Finally, the program creates a dialog box that inputs the log name and an array of Strings EFDefNames[] to display the exposed fields.

MainSample

The MainSample application is a top-level application that demonstrates how to call the runtime operations in the other sample files. The MainSample application represents one way of developing a simple application to work with the APIs.

This sample application does not use the Swing components found in the Step Processors and Launch Step Processor samples or the LogViewer sample. Instead, the sample relies on the command line input and the standard console for output.

Run the sample by entering a command similar to the following:

java LaunchSample username password <server name>:<port number>/<router instance name> [wfDefinition_filename | wfDefinition_filename output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

MainSample initializes a logger object and a session object, which will be used throughout the workflow process. MainSample instantiates the session object as an instance of the SessionHelper class and uses it to log onto a session. These two objects are then passed to sampleClass, which is an instance of the MainSample class.

The sampleClass object creates and launches a workflow definition, completes the steps, and writes history information to log files. To achieve this, the sampleClass object configures the workflow session with an instance of the SysConfigSample class. SysConfigSample creates a queue, an exposed field, and an index. To view details of workflow design and launch, see the WFDefinitionSample and LaunchSample classes, which are used by this sample.

The outer class sampleClass then invokes the runtime operations, which are performed by the following sample classes:

- RosterSample: demonstrates how to write the contents of a roster to a log file.
- StepProcessorSample: retrieves a step element from a queue, set the comments, displays the step information, and completes the step.
- QueueSample: displays contents of workflow queues, using QueueHelper.
- ProcessSample: illustrates how to log process information.
- LogSample: writes the VWLog record information to a file.

Note After you have reviewed, compiled, and run the samples, you will notice that some of the classes, like StepProcessorSample, are called more than once to perform similar operations.

RosterSample

This sample demonstrates how to write the contents of a roster to a log file. This sample uses the sample RosterHelper(VWSession, Logger) class to display roster information. Run the sample by entering a command similar to the following:

java RosterSample username password <server name>:<port number>/<router instance name> [output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The RosterSample class contains two methods: the main method main(String args[]) and the constructor method RosterSample(VWSession vwSession, Logger logger).

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default value for the log output file is **RosterSample.out**. The main method constructs and passes vwSession and Logger objects to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session and the logger to the sample's class constructor.

RosterSample(VWSession vwSession, Logger logger)

The constructor RosterSample(VWSession vwSession, Logger logger) performs common exception handling and invokes the rosterHelper.displayRosterContents() method with the following two lines of code:

```
// Create the roster helper object.

rosterHelper = new RosterHelper(vwSession, logger);

// Display the contents of the roster.

rosterHelper.displayRosterContents();
```

RosterHelper (helper class)

This is a helper class for some of the other samples. This sample demonstrates how to implement reusable roster-specific methods which can be accessed by other classes or applications.

Note The sample code assumes you created a session and supplied a user name and password with appropriate access privileges.

Methods

The following sections describe the public RosterHelper(VWSession, Logger) class methods.

public RosterHelper(VWSession vwSession, Logger logger)

The RosterHelper(VWSession, Logger) constructor sets variables m_vwSession and m_logger to point to the vwSession and logger object arguments, as shown below:

```
m_vwSession = vwSession;
m_logger = logger;
```

public void displayProcessInformation()

Displays the process information from the items in the default roster with the following processing steps:

Validate the session pointer:

```
   try {
if (m_vwSession == null){
   m_logger.log("Invalid session: <null> (displayProcessInformation)");
   return;
}
```

Use the m_vwSession.getRoster() method to get the roster object for the DefaultRoster:

```
vwRoster = m_vwSession.getRoster("DefaultRoster");
m_logger.log("Displaying process information for roster:  " + vwRoster.toString());

// Set the maximum number of items to 25, which requires less memory for each fetch than the default setting (50).

vwRoster.setBufferSize(25);

Construct a roster query object and query for all elements.  The fetch type is denoted by "VWFetchType.FETCH_TYPE_ROSTER_ELEMENT".

rQuery = vwRoster.createQuery(null, null, null, 0, null, null,
VWFetchType.FETCH_TYPE_ROSTER_ELEMENT);
```

Determine if any there any elements to retrieve with the query object, and get each roster element:

```
if (rQuery.hasNext()){
do
{

      // Get each roster element.

        rosterElement = (VWRosterElement)rQuery.next();
        m_logger.log("\nVWProcess information for roster element:\n");
```

Obtain a object process from the roster element's first work object:

```
workObject = rosterElement.fetchWorkObject(false, false);
if (workObject != null)
  process = workObject.fetchProcess();
if (process == null) continue;
```

Get workflow history & log key information from the process object.

Tip The mapId parameter is currently always -3 (PW version 4.0).

```
      wflHistory = process.fetchWorkflowHistory(-3);
```

Log the launch date of the workflow.

```
      m_logger.log("\tWorkflow Launch Date: " + wflHistory.getLaunchDate());
```

Log each of the step history element step IDs. Note the complementary use of the next() and hasNext() methods to control the loop:

```
                  m_logger.log("\n\tStep Histories:\n");

                  if (wflHistory.hasNext()) {

                  do {

                        stepHistory = wflHistory.next();
                        m_logger.log("\t\tStepId:  " +
                        stepHistory.getStepId());

                  } while (wflHistory.hasNext());

                  } else {
                    m_logger.log("\t\tNo Step Histories exist.");
                  }
```

Log the lock state for each remaining work objects in this process.

```
                  m_logger.log("\n\tWork Objects:\n");
                  if (process.hasNext()) {

                  do {
```

```
                        workObject = process.next();
                        m_logger.log("\t\tWork Object Id: " +
                        workObject.getWorkObjectNumber());

        // Get the lock state.

                        if (workObject.fetchLockedStatus() == 0)
                         m_logger.log("\t\t\tObject is not locked.");
                        else
                         m_logger.log("\t\t\tObject is locked.");

                    } while (process.hasNext());

                } else {
                   m_logger.log("\t\tNo Step Histories exist.");
                }

        } while (rQuery.hasNext());

        } else {
           m_logger.log("\t No roster elements, therefore can't get VWProcess information.");
        }
```

Handle any exceptions:

```
    }
catch (Exception ex)
{
  if (m_logger != null)
     m_logger.log(ex);
  else
     ex.printStackTrace();
  }
}
```

public void displayRosterContents()

Displays the contents of the "DefaultRoster" with the following steps:

Get the roster object for the DefaultRoster.

```
vwRoster = m_vwSession.getRoster("DefaultRoster");
```

Log the roster depth.

```
m_logger.log("Roster element count:  " + vwRoster.fetchCount());
```

Display the roster elements with a local method, as outlined below.

```
displayRosterElements(vwRoster);
```

Display the work objects with a local method.

displayWorkObjects(vwRoster);

public void displayRosterElements(VWRoster vwRoster) and public void displayWorkObjects(VWRoster vwRoster)

Displays the VWRosterElements in the specified VWRoster object with the following processing steps:

Construct a roster query object to query for all elements.

for displayRosterElements()

```
ElemQuery = vwRoster.createQuery(null, null, null, 0, null, null,
VWFetchType.FETCH_TYPE_ROSTER_ELEMENT);
```

for displayWorkObjects()

```
ElemQuery = vwRoster.createQuery(null, null, null, 0, null, null,
VWFetchType.FETCH_TYPE_WORKOBJECT);
```

Fetch the first element using the VWRosterQuery object and casting to the appropriate object type.

for displayRosterElements()

```
Element = (VWRosterElement)ElemQuery.next();
```

for displayWorkObjects()

```
Element = (VWWorkObject)ElemQuery.next();
```

Check to see if there are any elements:

```
if (Element == null)
   m_logger.log("\t Roster elements: none");
else
{
```

For each element, display the element information appropriately:

for displayWorkObjects()

```
  do
  {

  // Display the work object information.

     m_logger.displayWorkObjectInfo(vwWorkObject);
  }
  while ((vwWorkObject = (VWWorkObject)Element.next()) != null);
}
```

Additional code for displayWorkObjects() performs common error handling and cleanup.

for displayRosterElements()

```
// Initialize variables to hold the field names and each object value.

String[]    fieldNames = null;
Object      value = null;

    // For each roster element, iteratively process fields:

do {
m_logger.log("\t Roster element:");
```

This step and the remaining steps apply to Roster elements only. Display the system-defined field names and user-defined exposed field names supported by the roster element:

```
fieldNames = Element.getFieldNames();
if (fieldNames == null)
{
  m_logger.log("\t\t no Fields!");
}
else
{
  m_logger.log("\t\t Fields:");

      // For each field, iteratively display:

         for (int i = 0; i < fieldNames.length; i++) {

                  if (fieldNames[i] != null){

                           value = Element.getFieldValue(fieldNames[i]);

                  // Display the field names and their values:

                           m_logger.log("\t\t\t" + fieldNames[i] + "=" + value);

                  }

         }

}
```

Log specialized data available through VWRosterElement retrieval ("get") methods.

```
m_logger.log("\n\t\tOther Information:");
String bvalue = Element.getWorkObjectNumber();
m_logger.log("\t\t\t"+ "WorkObjectNumber" + "=" + bvalue);
String svalue = Element.getWorkObjectName();

m_logger.log("\t\t\t"+ "WorkObjectName" + "=" + svalue);
svalue = Element.getTag();
```

```
m_logger.log("\t\t\t"+ "Tag" + "=" + svalue);
int ivalue = Element.getServerLocation();
m_logger.log("\t\t\t"+ "CurrentServerLocation" + "=" + ivalue);
}
while ((rosterElement = (VWRosterElement)rElemQuery.next()) != null);

}
```

Additional code in this sample performs exception handling and cleanup.

StepProcessorSample

This sample demonstrates how to retrieve a step element from a queue, set the comments on the step element, display the step information, and complete the step. Run the sample by entering a command similar to the following:

java StepProcessorSample username password <server name>:<port number>/<router instance name> [output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The StepProcessorSample class contains two methods: the main(String args[]) method and the StepProcessorSample(VWSession vwSession, Logger logger, String queueName) method, which is the constructor.

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default value for the log output file is **StepProcessorSample.out**. The main method constructs and passes vwSession and Logger objects to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session, the logger, the user name, and the queue name to the constructor method.

StepProcessorSample(VWSession vwSession, Logger logger, String queueName)

The constructor StepProcessorSample(VWSession, Logger, String) performs common exception handling and retrieves a step element from a queue with the sample method queueHelper.getQueue(). It demonstrates setting the comments on the step element, displaying the step information, and dispatching the step with VWStepElement methods. The code to perform this is organized as follows:

Create an instance of the local sample QueueHelper class, and then create the object for the requested queue:

```
queueHelper = new QueueHelper(vwSession, logger);

        // get the requested queue

vwQueue = queueHelper.getQueue(queueName);
if (vwQueue != null)
{
```

Get a step element from the queue object:

```
  vwStepElement = queueHelper.getStepElement(vwQueue);
  if (vwStepElement != null)
  {
```

Lock the record whenever the step element changes:

```
        vwStepElement.doLock(true);
```

Set the comments:

```
        vwStepElement.setComment("This is the user's comment.");
```

Display the Step Processor information:

```
        logger.displayStepElementInfo(vwStepElement);
```

Dispatch the step:

```
      logger.log("Completing step: " + vwStepElement.getOperationName());
      vwStepElement.doDispatch();
  }
}
```

QueueSample

This sample demonstrates how to access and display queue contents, using the sample QueueHelper(VWSession, Logger) class to display queue information. Run the sample by entering a command similar to the following:

java QueueSample username password <server name>:<port number>/<router instance name> [queue_name] [output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The QueueSample class contains two methods: the main(String args[]) method and the QueueSample(VWSession vwSession, Logger logger, String queueName) method, which is the constructor.

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default values for the log output file and queue names are **QueueSample.out** and the first queue name in the system list of queue names, respectively. The main method constructs and passes vwSession and Logger objects to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session, the logger, and the queue name, which may be NULL.

QueueSample(VWSession vwSession, Logger logger, String queueName)

After the constructor QueueSample(VWSession, Logger, queueName) performs common exception handling, the method invokes methods from the sample queueHelper class and the APi class VWQueue to display queue contents. The code is organized as follows:

Create the roster helper object.

```
 queueHelper = new QueueHelper(vwSession, logger);
```

Get the queue object for the queueName passed in, or get first queue in the system list of queueNames.

```
if (queueName != null) { vwQueue = vwSession.getQueue(queueName);
} else {
    String[] queueNames = queueHelper.getQueueNames(false);
```

```
if (queueNames == null || queueNames.length == 0){
   logger.log("No queues found.");
   return;
} else {
```

Iterate through the array getting queues until first one with available elements is found.

```
        for (int i = 0; i < queueNames.length; i++){

                vwQueue = vwSession.getQueue(queueNames[i]);

                if (vwQueue != null) {
                  if (vwQueue.fetchCount() > 0)break;
                }
```

// Clear our reference.

```
                vwQueue = null;

        }

  }
}
```

// Ensure there is a VWQueue object.

```
if (vwQueue == null){

        logger.log("Unable to retrieve a queue!");
        return;

} else {
```

Display the contents of the queue:

```
    queueHelper.displayQueueContents(vwQueue);
}
```

Additional code in this sample manages common cleanup and error handling.

ProcessSample

This sample uses the sample RosterHelper(VWSession, Logger) class to display process information. Run the sample by entering a command similar to the following:

java ProcessSample username password <server name>:<port number>/<router instance name> [output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The ProcessSample class contains two methods: the main(String args[]) method and the ProcessSample(VWSession, Logger) method, which is the constructor.

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default value for the log output file is **ProcessSample.out**. The main method constructs and passes vwSession and Loggerobjects to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session and logger to the constructor method.

ProcessSample(VWSession vwSession, Logger logger)

The constructor ProcessSample(VWSession, Logger) performs common exception handling and invokes the RosterHelper.displayProcessInformation() method with the following two lines of code:

```
// create the roster helper object

rosterHelper = new RosterHelper(vwSession, logger);

// display the process information in the roster

rosterHelper.displayProcessInformation();
```

MilestoneHelper (helper file)

This is a helper class for other samples. The sample provides reusable getMilestoneDefinitions() and printMilestoneInfo() milestone-related methods for use by some of the sample classes, particularly the MileStoneHelper sample.

Note The sample code assumes you created a session and supplied a user name and password with appropriate access privileges. The MileStoneHelper constructor initializes local variables to point to its VWSession and Logger arguments as follows:

```
public MilestoneHelper(VWSession session, Logger logger) {

        m_vwSession = vwSession;
        m_logger = logger;

}
```

Methods

The following sections describe the public MileStoneHelper class methods.

VWMilestoneDefinition[] getMilestoneDefinitions(VWQueueElement vwQueueElement)

Milestone definitions are a part of the process information of a work object, and a queue element represents the workflow object in a queue. Although this queue element provides access to a fields in a queue without the expense of retrieving the work object, the API must use the original work object to obtain process information. The program fetches the passed in queue element's work object with VWQueueElement.fetchWorkObject(). The program gets process information from the work object with VWWorkObject.fetchProcess().

```
VWWorkObject workObj = vwQueueElement.fetchWorkObject(false, false);
VWProcess proc = workObj.fetchProcess();
```

The VWProcess.getMilestoneDefinitions() method retrieves the milestone definition information as an array of VWMileStoneDefinition objects. This object array is the value the method returns.

```
msd = proc.getMilestoneDefinitions();
```

printMilestoneInfo(VWMilestoneDefinition[] msd)

The milestone definitions obtained through the getMilestoneDefinitions method, in this class, contain milestone descriptions as a milestone name (String type), message (String type), and level (integer type). The program displays these values for each mileStone definition with logger.log(String).

```
// Iterate through all the milestones defined for the workflow
// and report each name, message, and level.
```

```
for (int i=0;i<msd.length;i++)
{
m_logger.log("\t\t" + msd[i].getName() + "\t\t" + msd[i].getMessage() + "\t" +
String.valueOf(msd[i].getLevel()));
}
```

MilestoneSample

This stand-alone sample demonstrates how to log the work object number and user information for locked work objects. Run the sample by entering a command similar to the following:

java MilestoneSample username password <server name>:<port number>/<router instance name> [queue_name] [output_file]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The MilestoneSample class contains two methods: the main(String args[]) method and the MilestoneSample(VWSession vwSession, Logger logger, String queueName) method, which is the constructor.

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default values for the log output file and queue names are **MilestoneSample.out** and the first queue name in the system list of queue names, respectively. The main method constructs and passes vwSession and Logger objects to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session, the logger, and the user name, and the name of queue to query, which may be null.

MilestoneSample(VWSession vwSession, Logger logger, String queueName)

The following code from the MilestoneSample constructor method displays the available milestones in the specified queue:

Instantiate the MilestoneHelper object, milestoneHelper. This class gets milestone defintions and prints milestone information.

milestoneHelper = new MilestoneHelper(vwSession, logger);

Instantiate the QueueHelper object, queueHelper. This class gets queue names and creates a VWQueue object.

queueHelper = new QueueHelper(vwSession, logger);

Get the specified queue object with vwSession.getQueue(), if the name is not null.

if (queueName != null){

```
            vwQueue = vwSession.getQueue(queueName);

}
else {
```

Otherwise, use the QueueHelper.getQueueNames() method, to test for the existence of queues.

```
        String[] queueNames = queueHelper.getQueueNames(false);
        if (queueNames == null || queueNames.length == 0)
        {

            logger.log("No queues found.");
            return;

        }
        else {
```

If queues are found, iteratively get queues until first queue with accessible elements is found. Create an instance of VWQueue with vwSession.getQueue(). vwQueue.fetchCount() returns the number of queues fetched.

```
            for (int i = 0; i < queueNames.length; i++){

            vwQueue = vwSession.getQueue(queueNames[i]);
            if (vwQueue != null){

                if (vwQueue.fetchCount() > 0)
                break;

            }

            // Clear the queue pointer reference.

                vwQueue = null;

        }

}
```

If a queue object is retrieved, retrieve the first element and create a query object for the queue object with the API method VWQueue.createQuery().

```
qQuery = vwQueue.createQuery(null, null, null, 0, null, null,
VWFetchType.FETCH_TYPE_QUEUE_ELEMENT);
```

Attempt to retrieve the query object's first queue element.

```
vwQueueElement = (VWQueueElement)qQuery.next();
if (vwQueueElement == null){

            logger.log("\t Queue elements: none");

}
```

else {

If a queue element is retrieved, log its workflow name with the logger.log() method. Obtain the name from the vwQueueElement.getWorkflowName() method. Get milestone definitions for every element, and print them using the MilestoneHelper methods getMilestoneDefinitions() and printMilestoneInfo().

do {

```
logger.log("\tWorkflow " + vwQueueElement.getWorkflowName());
VWMilestoneDefinition[] msd =
milestoneHelper.getMilestoneDefinitions(vwQueueElement);
milestoneHelper.printMilestoneInfo(msd);
```

} while ((vwQueueElement = (VWQueueElement)qQuery.next()) != null);

Additional code in this method performs common cleanup and error handling.

OperationsHelper (helper class)

This is a helper class for other samples. The sample provides system step-related methods for use by some of the sample classes, particularly the OperationsSample. The methods in this sample class modify and validate workflows.

Note The sample code assumes you created a session and supplied a user name and password with appropriate access privileges. The OperationsHelper constructor initializes local variables to point to its SessionHelper and Logger arguments as follows:

```
public OperationsHelper(VWSession session, Logger logger) {

        m_vwSession = vwSession;
        m_logger = logger;

}
```

Methods

The following sections describe the public OperationsHelper class methods.

VWOperationDefinition getOperationDefinition(String operationName, VWQueue theQueue)

Retrieve the definition of an operation as follows:

Create default VWQueueDefinition and VWOperationDefinition objects. The VWQueueDefinition object holds the definition of the passed queue, and the VWOperationDefinition object stores the return value.

```
VWQueueDefinition vwQueueDef = null;
VWOperationDefinition vwOpDef = null;
```

Retrieve the operation definition from the queue definition with VWQueueDefinition.getOperation() and the passed in operation name, after setting the VWQueueDefinition object with VWQueue.fetchQueueDefinition(). Return the operation definition.

```
vwQueueDef = theQueue.fetchQueueDefinition();
vwOpDef = vwQueueDef.getOperation(operationName);

// . . . ( code for common messge and error handling ) . . .

return vwOpDef;
```

printOperationDetails(VWOperationDefinition vwOpDef)

Logs VWOperationDefinition object information as follows:

Log the name and description of the passed in operation definition object with the sample Logger.log method.

```
m_logger.log("\tName: " + vwOpDef.getName());
m_logger.log("\t\tDescription: " + vwOpDef.getDescription());
```

Print the parameter information of the passed in operation definition with the sample OperationsHelper.printOperationParameters() method.

```
printOperationParameters(vwOpDef);
```

Additional code in this method performs common error and message handling.

printOperationParameters(VWOperationDefinition vwOpDef)

Log the parameter information of the passed in operation definition as follows:

Initialize an array of VWParameterDefinition objects.

```
VWParameterDefinition[] vwPD = null;
vwPD = vwOpDef.getParameterDefinitions();
```

Log the name, datatype, and value of each operation with VWParameterDefinition retrieval ("get") methods.

```
m_logger.log("\t\tName\t\tData Type\tValue");
for (int i=0;i<vwPD.length;i++){

        m_logger.log("\t\t" + vwPD[i].getName() + "\t\t" +
        VWFieldType.getLocalizedString(vwPD[i].getDataType()) + "\t" + vwPD[i].getValue());

}
```

Additional code in this method performs common error and message handling.

OperationsSample

This stand-alone sample demonstrates how to use the API classes to manage queue operations. Run the sample by entering a command similar to the following:

java OperationsSample username password <server name>:<port number>/<router instance name> [queue_name] [output_file]

If no queue name is specified, the program uses the first queue with elements in the list of queueNames. If do not specify an output file name, the sample uses **OperationSample.out** as the default location.

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

The OperationsSample code creates an instance of the SessionHelper class (sessionHelper) and the Logger class (logger). The sessionHelper.logon() method logs on to a workflow session and returns a VWSession object, vwSession. The main method then passes the vwSession object, the logger object, and the queue name, which may be null, to the sample constructor.

The following code from the OperationsSample constructor method configures and reports on queue operations.

Get a valid queue name

Create default operations and queue helper objects.

```
operationsHelper = new OperationsHelper(vwSession, logger);
queueHelper = new QueueHelper(vwSession, logger);
```

Get the specified queue object with vwSession.getQueue(), if the name was not NULL.

```
if (queueName != null){
        vwQueue = vwSession.getQueue(queueName);
} else { // ( . . . continued below)
```

Otherwise, use the QueueHelper.getQueueNames() method, to test for exsiting queues.

```
        String[] queueNames = queueHelper.getQueueNames(false);
        if (queueNames == null || queueNames.length == 0)
        {
            logger.log("No queues found.");
            return;
        } else {
            vwQueue = vwSession.getQueue(queueNames[1]);
```

```
}

    // Make sure the program has created a VWQueue object.

if (vwQueue == null){

    logger.log("Unable to retrieve a queue!");
    return;

} else {
```

Add an operation

```
    logger.log("Phase 1 : Configuration\n");
```

Fetch a VWQueueDefinition object with VWQueueDefinition and initialize a VWOperationDefinition object.

```
    VWQueueDefinition vwQueueDef = null;
    VWOperationDefinition opDef = null;
    vwQueueDef = vwQueue.fetchQueueDefinition();
```

Create a new operation definition with the VWQueueDefinition.createOperation() method and add some parameters to it.

```
    opDef = vwQueueDef.createOperation("Sample Operation");
    logger.log("Created new operation definition");

    // Add boolean parameter, sent by the operation, and not an array.

    opDef.createParameter("BoolParameter", 2, 4, false);

    // Add an integer parameter, returned to the operation, and not an array.

    opDef.createParameter("IntParameter", 1, 1, false);
    opDef.setDescription("Created by OperationsSample example application");
    logger.log("Configured operation definition");
```

Update the system configuration queue definition, using the current queue copy and VWSystemConfiguration.updateQueueDefinition().

```
    sysConfig = vwSession.fetchSystemConfiguration();
    sysConfig.updateQueueDefinition(vwQueueDef);
```

Commit changes. The update is not complete until it is committed.

```
    String[] errors = sysConfig.commit();
    if (errors != null)

        logger.log("Errors: ", errors);

    else
```

```
        logger.log("Committed configuration changes.");
```

Report on operations for the current queue

Fetch a list of operations defined on this queue. Get each operation information and print it with the sample helper methods operationsHelper.getOperationDefinition() and printOperationDetails().

```
    String[] operationNames = vwQueue.fetchOperationNames();

    VWOperationDefinition vwOpDef = null;
    for (int i=0;i<operationNames.length;i++){

    // get configuration information about each defined operation

        vwOpDef = operationsHelper.getOperationDefinition(operationNames[i],
        vwQueue);
        operationsHelper.printOperationDetails(vwOpDef);

    }

}
```

Additional OperationsSample code performs common message and error handling.

SystemStepHelper (helper class)

This is a helper class for other samples. This sample provides system step-related methods to some of the sample classes, particularly the SystemStepSample. The methods in this sample class assist with adding steps to workflows and validating workflows.

Note The sample code assumes you created a session and supplied a user name and password with appropriate access privileges. The SystemStepHelper constructor initializes local variables to point to the VWSession and Logger arguments as follows:

```
public SystemStepHelper(VWSession session, Logger logger) {

        m_vwSession = vwSession;
        m_logger = logger;

}
```

Methods

The following sections describe the public SystemStepHelper class methods.

VWCompoundStepDefinition addCompoundStep(VWMapDefinition mapDef, String stepName)

Create a VWCompoundStepDefinition object as a compound step definition with VWMapDefinition.createCompoundStep() and the passed in stepname. Get the ID of the compound step definition with VWCompoundStepDefinition.getStepId().

```
newStepDef = mapDef.createCompoundStep(stepName);
nStepId = newStepDef.getStepId();
```

Set the step description and map location with the step id and VWCompoundStepDefinition storage ("set") methods.

```
newStepDef.setDescription("This is the description for system step" + nStepId + ".");
newStepDef.setLocation(new java.awt.Point(nStepId * 100, 150));
```

Additional code in this method performs common message and error handling.

VWStepDefinition addStep(VWMapDefinition mapDef, String stepName, String queueName)

Create a default step with the passed in name and get the new step Id with VWMapDefinition methods.

```
newStepDef = mapDef.createStep(stepName);
nStepId = newStepDef.getStepId();
```

Create integer and string parameters for the new step definition with the VWStepDefinition.createParameter() method.

```
newStepDef.createParameter( "Field1_Integer", VWModeType.MODE_TYPE_IN, "99",
VWFieldType.FIELD_TYPE_INT, false );
newStepDef.createParameter( "Field2_String", VWModeType.MODE_TYPE_OUT, "Field2_String",
VWFieldType.FIELD_TYPE_STRING, true );
```

Assign the new step the passed in queue name with VWStepDefinition.newStepDef.setQueueName().

```
newStepDef.setQueueName(queueName);
```

Store the step description and map location in the new VWStepDefinition object with its storage ("get") methods. The map location determines where it will graphically display in Designer

```
newStepDef.setDescription("This is the description for step" + nStepId + ".");
newStepDef.setLocation(new java.awt.Point(nStepId * 100, 150));
```

Additional code in this method performs common message and error handling.

boolean validate(VWWorkflowDefinition workflowDef, VWSession vwSession)

Initialize an array of VWValidationError objects and a variable to hold the return value for the method.

```
VWValidationError[] validationErrors = null;
boolean bSuccess = false;
```

Validate the passed in workflow by setting the VWValidationError array with the first argument's VWWorkflowDefinition.validate() method; then test the array.

```
validationErrors = workflowDef.validate(vwSession, false);
if (validationErrors != null){

        m_logger.log("\nThe following validation errors occurred: ", validationErrors);

} else {

        m_logger.log("\nValidation was successful.\n");
        bSuccess = true;

}
```

Additional code in this method performs common message and error handling.

SystemStepSample

This stand-alone sample demonstrates how to create a workflow, set some properties, add a system step, and launch the workflow. Run the sample by entering a command similar to the following:

java SystemStepSample username password <server name>:<port number>/<router instance name> [queue_name] [output_file]

If no queue name is specified, the program does not associate a queue with the workflow. A workflow without a queue cannot include a Step Processor. If no output file is specified, the name **SystemStepSample.out** is used.

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The SystemStepSample class contains two methods: the main(String args[]) method and the SystemStepSample(VWSession vwSession, Logger logger, String queueName) method, which is the constructor.

main(String args[])

The SystemStepSample main method code creates an instance of the SessionHelper class (sessionHelper) and the Logger class (logger). The sessionHelper.logon() method logs on to a workflow session and returns a VWSession object, vwSession. The main method then passes the vwSession object, the logger object, and the queue name to the sample constructor.

SystemStepSample(VWSession vwSession, Logger logger, String queueName)

The SystemStepSample constructor method shows how to create a workflow with a system step. The sample performs these operations in the general order shown below.

Instantiate a SystemStepHelper object with the vwSession and logger object arguments. This SystemStepHelper object will later assist the current object with its addStep, addCompoundStep, and validate methods.

SystemStepHelper systemStepHelper = new SystemStepHelper(vwSession, logger);

Create the VWWorkflowDefinition object, workflowDef.

```
VWWorkflowDefinition workflowDef = null;
// ( additional declaration code . . . )
workflowDef = new VWWorkflowDefinition();
```

Set some workflow definition properties.

```
workflowDef.setSubject("\"This is the System Step sample workflow\"");
workflowDef.setDescription("This is a workflow generated by the System Step Sample");
workflowDef.setName("System Step Workflow");
```

Define some workflow definition fields with VWWorkflowDefinition.createFieldUsingString().

```
workflowDef.createFieldUsingString("Field1_Integer", "99", VWFieldType.FIELD_TYPE_INT, false);
workflowDef.createFieldUsingString("Field2_String", "{\"\"}", VWFieldType.FIELD_TYPE_STRING,
true);
```

Get a VWMapDefinition object as the default map with VWWorkflowDefinition.getMainMap(), and set the map description property.

```
mapDef = workflowDef.getMainMap();
mapDef.setDescription("This is the sample workflow map");
```

Get a VWMapNode object as the launch step with VWWorkflowDefinition.getStartStep(), and set the launch step description map property.

```
currentStepDef = mapDef.getStartStep();
currentStepDef.setDescription("This is the description for the launch step.");
```

Define the first step after the launch as a system step. The program must first set another VWMapNode object to hold the new launch step.

```
precedingStepDef = currentStepDef;
```

Construct, then set a compound step ("Step 1") with the sample helper class method SystemStepHelper.addCompoundStep ().

```
VWCompoundStepDefinition sysStepDef = null;
sysStepDef = systemStepHelper.addCompoundStep(mapDef, "Step 1");
```

Create an assignment instruction which initializes system step field "Field1_Integer" to 42.

```
sysStepDef.createAssignInstruction(new String[][] {{"Field1_Integer","42"}} );
```

Create a BeginTimer instruction with the VWCompoundStepDefinition.createBeginTimerInstruction() method. The arguments name the Begintimer "Timer 1", set it to expire 15 minutes after reaching this step, and set its expiration to branch to the system workflow instruction "Terminate". This timer is "non-preemptive", which means it will wait until the current step completes before it terminates.

```
sysStepDef.createBeginTimerInstruction("\"Timer 1\"", "addminutes(systemtime(),15)", "Terminate",
null);
```

Reset the value of the current step definition object variable to the compound system step the program just defined. This step remains defined in the "precedingStepDef" object.

```
currentStepDef = sysStepDef;
```

Use the"precedingStepDef" object to create a route from the preceeding (launch) step to the new step (step 1). Note that the createRoute() argument specifies the new step ID by invoking VWNode.getStepId().

```
precedingStepDef.createRoute(currentStepDef.getStepId());
```

Create three more (non-compound) steps in the workflow (main) map with the sample class method systemStepHelper.addStep(). If there is no associated queue name, a null value for queueName is valid. VWNode.createRoute() establishes the route from step to step.

```
for (int i = 0; i < 3; i++){

        currentStepDef = systemStepHelper.addStep(mapDef, "Step" + i, queueName);
        if (currentStepDef != null){

                        precedingStepDef.createRoute(currentStepDef.getStepId());

        // Reset the preceeding step lag variable.

                        precedingStepDef = currentStepDef;

        }

}
```

Create a fifth step that terminates "Timer 1", initiated by "Step 1", above. The procedure is much the same as the one that created BeginTimer "Step 1", the previous compound step. The method uses VWNode.createEndTimerInstruction() in place of VWNode.createBeginTimerInstruction(), as follows:

```
sysStepDef = null;
sysStepDef = systemStepHelper.addCompoundStep(mapDef, "Step 5")
sysStepDef.createEndTimerInstruction("\"Timer 1\"");
currentStepDef = sysStepDef;

// Create a route from the preceeding step to the new one.

precedingStepDef.createRoute(currentStepDef.getStepId());
```

The program validates the new workflow definition with the sample method systemStepHelper.validate().

```
if (systemStepHelper.validate(workflowDef, vwSession)){ // ( . . . continue processing)}
```

Write the local workflow definition file, which contains an XML representation of the workflow definition. (To see a graphical representation of the workflow, use Designer to open this file.)

```
workflowDef.writeToFile("SystemStepSample.pep");
```

Transfer the workflow definition with VWSession.transfer().

```
VWTransferResult transferResult = vwSession.transfer(workflowDef, "uniqueid", false, true);
```

Use the resulting VWTransferResult object to show whether the transfer was successful or not with the

following code:

```
if (transferResult.success()){

        vwVersion = transferResult.getVersion();
        logger.log("The transfer was successful.");

}else { ( . . . Display the transfer errors ) }
```

Launch the workflow: create a step Element with vwSession.createWorkflow(vwVersion), optionally set a comment with VWStepElement.setComment(), and dispatch the workflow with VWStepElement.doDispatch().

```
VWStepElement launchStep = vwSession.createWorkflow(vwVersion);
launchStep.setComment("This is the System Step sample launch step comment");
launchStep.doDispatch();
```

Additional code in SystemStepSample manages common messages and errors.

UserInfoSample

This sample demonstrates how to set email notification for all users added to the workflow system. Run the sample by entering a command similar to the following:

java UserInfoSample username password <server name>:<port number>/<router instance name> email_suffix

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

The main method displays a usage line if you supply less than four arguments; otherwise, the method passes the arguments to the constructor for the sample. The constructor UserInfoSample(String user, String pw, String router, String email_suffix) performs common exception handling and sets user information properties for email notification. This sample operates in the general order listed below.

Create an instance of the session object. Create the security list object from the session object. In the code below, each fetch to the security list object returns 1000 elements, and the false argument causes the list to consist of users, only, to exclude groups.

```
session = new VWSession(user, pw, router);
VWSecurityList sl = session.fetchUsers(1000, false);

while (sl.hasNext()){
```

Create an instance of a user information object from the session and an element of the security list object, using VWSession.fetchUserInfo(String). The returned security list object is cast to a String for use as an argument.

```
    VWUserInfo ui = session.fetchUserInfo((String)sl.next());
```

Print the email address and set it with the VWUserInfo.setEMailAddress(String) method.

```
    System.out.println(ui.getName() + email_suffix);
    ui.setEMailAddress(ui.getName() + email_suffix);
```

This step occurs within a while loop for instructional reasons. In production, the same value should be set one time before the while loop to reduce runtime redundancy and to perform more efficiently.

Create a combination notification flag for this user to be notified by email, as various events occur. The flags are combined by performing a bitwise or of each value to be included.

```
    int nf = VWUserInfo.NOTIFICATION_STEP_EXPIRED_DEADLINE |
    VWUserInfo.NOTIFICATION_STEP_NEW_ASSIGNMENT |
    VWUserInfo.NOTIFICATION_STEP_REMINDERS |
    VWUserInfo.NOTIFICATION_TRACKER_EXPIRED_DEADLINE |
    VWUserInfo.NOTIFICATION_TRACKER_NEW_ASSIGNMENT |
    VWUserInfo.NOTIFICATION_TRACKER_WORKFLOW_EXCEPTION;
```

Set the notification flags and save the user information with VWUserInfo methods:

```
        ui.setNotificationFlags(nf);
        ui.save();
```

}

Additional code in this sample performs common error handling and cleanup.

WorkPerformerSample

This sample demonstrates how to retrieve, modify, and complete a step using the VWWorkObject class. Run the sample by entering a command similar to the following:

java WorkPerformerSample username password <server name>:<port number>/<router instance name> queueName [output_filename]

Note For a detailed explanation of the command line, see the Run the sample application section of the Run the Unmodified Samples topic.

Methods

The WorkPerformerSample class contains five methods: main(String args[]), run( ), processQueue(VWQueue vwQueue), processWork(VWWorkObject workObject), and constructor WorkPerformerSample(VWSession vwSession, Logger logger, String queueName).

main(String args[])

The main method uses common techniques for validating and defaulting argument values. The default value for the log output file is **WorkPerformerSample.out**. The main() method constructs and passes vwSession and Logger objects to the sample constructor. Main() handles the login and logoff for the session with the login() and logoff() methods of the sample SessionHelper class. It provides workflow logging with an instance of the sample Logger class. The main method passes the session, the logger, the user name, and the queue name to the constructor method.

WorkPerformerSample(VWSession vwSession, Logger logger, String queueName)

The constructor WorkPerformerSample(VWSession, Logger, String) performs common exception handling and retrieves a VWQueue object of class scope with the sample method queueHelper.getQueue(). A class-scope boolean variable (m_bDone) is initialized outside the constructor. This variable serves as a control flag for terminating the run program.

A character variable to hold input keystrokes from the user is initialized within the constructor. A thread is started, and a message tells the user to hit the enter key to exit. The code captures user keystroke input in a loop and tests whether a newline character was entered.

```
// Start the process thread.

Thread thread = new Thread(this, "WorkPerformerSample");
thread.start();

// Wait for a keystroke.

System.out.print("Hit Enter key to exit:");
```

```
while (!m_bDone){

        try {

                nCh = System.in.read();
                if (nCh < 0 || (char)nCh == '\n')

                        m_bDone = true;

        }catch(java.io.IOException e)
        {

                m_bDone = true;

        }

}
```

Note that control exits this loop when a new line is entered. Control does not pass to the termination code until the thread that was started terminates, as shown below:

```
System.out.print("Finishing processing - please wait.");
while (thread.isAlive());
```

The termination code in main() reports the completion of the program to the user.

```
finally {
if (m_logger != null)

        m_logger.logAndDisplay("~ WorkPerformerSample execution complete.\n");

}
```

The additional constructor code that follows the above loop performs routine messaging and error handling.

After the program invokes the Java method thread.start(), the code in WorkPerformerSample.run() activates. This is the run process.

void run()

This method executes a simple loop that tests the boolean "done" flag (m_bDone) and searches the queue every 30 seconds. Other included code performs routine error handling. Note that if the mbDone flag is true, this method and the thread it runs in terminate. Otherwise, the m_bDone flag does not allow the parent object to terminate.

```
while (!m_bDone)
{

        // Search the queue.

        processQueue(m_vwQueue);
```

```
            // Pause 30 seconds.
            if (!m_bDone)
                Thread.sleep(30000);
}
```

private void processQueue(VWQueue vwQueue)

Initialize null VWQueueQuery and VWWorkObject objects:

```
VWQueueQuery qQuery = null;
VWWorkObject workObject = null;
```

Set a buffer size for querying the queue that was passed in and construct a query object.

```
vwQueue.setBufferSize(25);
qQuery = vwQueue.createQuery(null, null, null, 0, null, null,
VWFetchType.FETCH_TYPE_WORKOBJECT);
```

After testing for a valid query object, query for all step elements. For each element found, process it with the processWork() method.

```
while (qQuery.hasNext()) {
    // Get each work object and process it.
    workObject = (VWWorkObject)qQuery.next();
    if (workObject != null)
            processWork(workObject);
}
```

private void processWork(VWWorkObject workObject)

Each work object obtained from the query of the user-selected queue is processed in this method by work object methods VWWorkObject.getFieldValue(), VWWorkObject.setFieldValue(), and VWWorkObject.doDispatch().

Before processing a workObject, it must be locked with the VWWorkObject.doLock(boolean) method.

```
workObject.doLock(true);
```

The program tests each work object for the correct name of any fields to be retrieved or set, using VWWorkObject.hasFieldName(String). Two examples are as follows:

```
if (workObject.hasFieldName("Title")) {
        title = (String)workObject.getFieldValue("Title");
```

```
}

// Set the comment.

if (workObject.hasFieldName("F_Comment")) {

        workObject.setFieldValue("F_Comment", "Processed by WorkPerformer",
        true);

}
```

After retrieving and setting field values in the work object, the work object is dispatched with the doDispatch() method. A work object should be locked when it is dispatched.

```
workObject.doDispatch();
```

Modify Core Script Files

This topic describes how to modify, in general terms, the core script files used by the default HTML Step Processors and Launch Step Processors. Unlike the other topics detailing the files in the \WF_Html_toolkit\core directory, this file lists the constructors and available method for the client-side object.

Caution You can add new functions or extend the existing function in these files; however, you must not delete or alter the existing functions.

step.js

The **step.js** file defines and declares methods for the creating and working with step element (PW_StepElement), step information (PW_StepInfo), step parameter (PW_StepParameter), milestone (PW_Milestone), and work group (PW_WorkGroup) objects.

This topic summarizes the methods supported for each Javascript object. If you want more detailed objects reference information, review the Javascript code contained in the file for the constructor and method parameter information.

PW_StepElement (Step Element) Object

The PW_StepElement object supports the following methods:

- addAttachment() - add an attachment object to a step element
- getAttachment() - retrieve a specific attachment
- getAttachmentCount() - determine the number of attachments on a step element
- getResponse() - check the response array for an index value
- setComment() - set the comment for the step element
- setSubject() - set the subject for the step element
- setUserInfoList() - add user information to an array
- toXML() - create an XML string that represents the step element

PW_StepInfo (Step Information) Object

The PW_StepInfo object does not support any methods.

PW_StepParameter (Step Parameter) Object

The PW_StepParameter object supports the following methods:

- getDescription() - returns the parameter description
- setDirty() - modies boolean to indicate the parameter needs to be saved to the server.
- toString() - returns the parameter name

- toXML() - creates an XML string, using the step information, to send from the client to the server.

PW_Milestone (Milestone) Object

The PW_Milestone object supports the following methods:

- getName() - retrieve the milestone name.
- getMessage() - retrieve the message associated with the milestone.
- getReached() - get the information about the milestone.
- toXML() - creates an XML string, using the milestone information, to send from the client to the server

PW_WorkGroup (Work Group) Object

The PW_WorkGroup object supports the following methods:

- getDescription() - returns the work group description
- setDirty() - modies boolean to indicate the parameter needs to be saved to the server.
- toString() - return the work group name
- toXML() - create an XML string, using the work group information, to send from the client to the server.

---

att.js

The **att.js** file defines and declares methods for the creating and working with attachment (PW_Attachment) objects. Review the Javascript code contained in the file for more information about constructor information and method parameter information.

Because the attachments are stored in either Content Services or Image Services libraries, the implemented functions rely on the client-side Javascript IDMWSC_Library object, which is supplied by the Panagon Web Services, to create and work with attachment objects. Refer to the Referenced Panagon Web Services Files for a brief description of the available client-side API objects, or refer to Locate the Panagon Web Services Toolkit Help for more information on using the Panagon Web Services components.

PW_Attachment (Attachment) Object

The PW_Attachment object supports the following methods:

- assign() - assign an attachment a specific place in an array
- deleteIDMObj() - delete an attachment from a specific place in an array
- getDescription() - return an attachment description
- getIconFile() - return a string to locate the icon associated with the object type
- getIDMAttachment() - retrieve an array from a specific location in an array

- getIDMAttachmentCount() - determine the number of attachments in an array
- getToolTip() - return a description of well-known object types
- insert() - put an attachment into an array
- setDirty() - set a flag to true to indicate that the attachment parameter was modified and it needs to be saved to the server
- setReadOnly() - prevents UI from changing the attachment parameter value
- toString() - get the attachment name
- toXML() - create and XML string the describes the attachment object
- unassignIDMObj() - unassign an attachment at a given location in an array