Sun Java™ System

Identity Manager 7.1 Deployment Tools

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820-0820-10

Contents

List of Figures

List of Tables

Preface

This *Sun Java™ System Identity Manager Deployment Tools* publication provides reference and procedural information to help you use different Identity Manager deployment tools.

Who Should Use This Book

Sun Java™ System Identity Manager Deployment Tools was designed for deployers and administrators who will create and update workflows, views, rules, system configurations and other configuration files necessary to customize Identity Manager for a customer installation during different phases of product deployment.

Deployers should have a background in programming and should be comfortable with XML, Java, Emacs and/or IDEs such as Eclipse or NetBeans.

Administrators do not need a programming background, but should be highly skilled in one or more resource domains such as LDAP, Active Directory, or SQL.

How This Book Is Organized

Identity Manager Deployment Tools is organized into these chapters:

- Chapter 1, "Using the Identity Manager IDE" — Introduces the Identity Manager Integrated Development Environment (Identity Manager IDE), and provides instructions for installing this application.
- Chapter 2, "Working with Rules" — Describes functions that typically consist of XML, or alternatively JavaScript, in an XPRESS wrapper. Rules provide a mechanism for storing frequently used XPRESS logic or static variables for easy reuse within forms, workflows, and roles.
- Chapter 3, "Working with Variable Namespaces" — Provides an overview of common Identity Manager tasks or processes, how they are typically used, and the namespace in which they run.
- Chapter 4, "Developing Adapters" — Describes how to create custom Identity Manager resource adapters that are tailored to your company or customers.
- Chapter 5, "Working with Firewalls or Proxy Servers" — Describes how Identity Manager uses Uniform Resource Locators (URLs) and how to obtain accurate URL data when firewalls or proxy servers are in place
- Chapter 6, "Configuring Dictionary Support" — Describes how to configure and set up dictionary support in Identity Manager.
- Chapter 7, "Using SPML 1.0 with Identity Manager Web Services" — Provides details about using the SOAP-based Web service interface provided by the Identity Manager server. Describes the SPML 1.0 classes used to format request messages and parse response messages.
- Chapter 8, "Using SPML 2.0 with Identity Manager Web Services" — Provides details about using the SOAP-based Web service interface provided by the Identity Manager server. Describes the SPML 2.0 classes used to format request messages and parse response messages.
- Appendix A, "Using the Business Process Editor" — Describes the Identity Manager Business Process Editor (BPE), and provides instructions for using this application.

Conventions Used in This Book

The tables in this section describe the conventions used in this book including:

- Typographic Conventions
- Symbols
- Shell Prompts

Typographic Conventions

The following table describes the typographic conventions used in this book.

Table 1 Typographic Conventions

Typeface	Meaning	Examples
`AaBbCc123` (Monospace)	API and language elements, HTML tags, Web site URLs, command names, file names, directory path names, on-screen computer output, sample code.	Edit your `.login` file. Use `ls -a` to list all files. `% You have mail.`
`AaBbCc123` (Monospace bold)	What you type, when contrasted with onscreen computer output.	`% su` `Password:`
AaBbCc123 (Italic)	Book titles, new terms, words to be emphasized. A placeholder in a command or path name to be replaced with a real name or value.	Read Chapter 6 in the *User's Guide*. These are called *class* options. Do *not* save the file. The file is located in the *install-dir*`/bin` directory.

Symbols

The following table describes the symbol conventions used in this book.

Table 2 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional command options.	`ls [-l]`	The `-l` option is not required.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.

Table 2 Symbol Conventions*(Continued)*

Symbol	Description	Example	Meaning
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
>	Indicates menu item selection in a graphical user interface.	File > New > Templates	From the File menu, choose New. From the New submenu, choose Templates.

Shell Prompts

The following table describes the shell prompts used in this book.

Table 3 Shell Prompts

Shell	Prompt
C shell on UNIX or Linux	*machine-name*%
C shell superuser on UNIX or Linux	*machine-name*#
Bourne shell and Korn shell on UNIX or Linux	$
Bourne shell and Korn shell superuser on UNIX or Linux	#
Windows command line	C:\

Related Documentation and Help

Sun Microsystems provides additional documentation and information to help you install, use, and configure Identity Manager:

- *Identity Manager Installation*: Step-by-step instructions and reference information to help you install and configure Identity Manager and associated software.
- *Identity Manager Upgrade*: Step-by-step instructions and reference information to help you upgrade and configure Identity Manager and associated software.
- *Identity Manager Administration*: Procedures, tutorials, and examples that describe how to use Identity Manager to provide secure user access to your enterprise information systems.
- *Identity Manager Technical Deployment Overview*: Conceptual overview of the Identity Manager product (including object architectures) with an introduction to basic product components.
- *Identity Manager Workflows, Forms, and Views*: Reference and procedural information that describes how to use the Identity Manager workflows, forms, and views — including information about the tools you need to customize these objects.
- *Identity Manager Resources Reference*: Reference and procedural information that describes how to load and synchronize account information from a resource into Sun Java™ System Identity Manager.
- *Identity Manager Tuning, Troubleshooting, and Error Messages*: Reference and procedural information that provides guidance for tuning Sun Java™ System Identity Manager, provide instructions for tracing and troubleshooting problems, and describe the error messages and exceptions you might encounter as you work with the product.
- *Identity Manager Service Provider Edition Deployment*: Reference and procedural information that describes how to plan and implement Sun Java™ System Identity Manager Service Provider Edition.
- *Identity Manager Help*: Online guidance and information that offer complete procedural, reference, and terminology information about Identity Manager. You can access help by clicking the Help link from the Identity Manager menu bar. Guidance (field-specific information) is available on key fields.

Accessing Sun Resources Online

For product downloads, professional services, patches and support, and additional developer information, go to the following:

- Download Center
  http://wwws.sun.com/software/download/
- Professional Services
  http://www.sun.com/service/sunps/sunone/index.html
- Sun Enterprise Services, Solaris Patches, and Support
  http://sunsolve.sun.com/
- Developer Information
  http://developers.sun.com/prodtech/index.html

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in the product documentation, contact customer support using one of the following mechanisms:

- The online support Web site at http://www.sun.com/service/online/us
- The telephone dispatch number associated with your maintenance contract

Related Third-Party Web Site References

Sun is not responsible for the availability of third-party Web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions.

To share your comments, go to http://docs.sun.com and click Send Comments. In the online form, provide the document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the book or at the top of the document.

For example, the title of this book is *Sun Java™ System Identity Manager Deployment Tools*, and the part number is 820-0820-10.

Chapter 1

Using the Identity Manager IDE

The Identity Manager Integrated Development Environment (Identity Manager IDE) is Java application that enables you to view, customize, and debug Sun Java™ System Identity Manager (Identity Manager) objects in your deployment.

This chapter provides instructions for using the Identity Manager IDE. The information in this chapter is organized as follows:

Overview

The Identity Manager IDE provides a graphical and XML-based view of objects in your environment, and you can use this application to perform the following tasks:

- Create projects that are associated with specific repositories
- View, create, and edit configuration objects, email templates, forms, generic objects, libraries, rules, workflow processes, and workflow subprocesses
- Download objects from and upload objects to the repository
- Debug forms, rules, and workflows

The rest of this section provides a general, high-level overview of the Identity Manager IDE.

Major Features

Major features provided by the Identity Manager IDE include:

- Integrated Explorer window that allows project, directory-based, or run-time views of a project
- Identity Manager IDE projects are integrated with a Configuration Build Environment (CBE)
- Action menus for document modification
- Custom editors, including:
  - Object property sheets and graphical value editors for enumerating XML object properties and editing basic object types, XPRESS, and XML objects without typing XML
  - Drag and drop palette for adding approvals, workflow services, workflow tasks, users, and so forth to XML source without typing XML
  - Registered `waveset.dtd` definition file that enables syntax highlighting and auto-completion for XML elements and attributes
- Integrated debugger for forms, rules, and workflows
- Rule tester for verifying standalone and library rules
- Form tester for testing forms in an external browser

- Checkout View feature allows you to check out, modify, and check in Identity Manager views (such as a user view).
- CVS integration

How Identity Manager IDE Compares to BPE

The Identity Manager IDE is a fully integrated NetBeans plugin that was designed to replace Identity Manager's Business Process Editor (BPE) application. When compared with the BPE, Identity Manager IDE offers the following advantages:

- Provides closer alignment with deployment best practices, including:
  - Modifying objects outside of the repository
  - Using common deployment tools such as Ant and CVS
- Provides platform support for editors, palette, debugger, and navigation
- Reduces open-endedness by providing code completion for XML elements and attributes
- Provides contextual information, with integrated tools to facilitate editing

NOTE The BPE is still supported for backward compatibility. For information about using the BPE, see Appendix A, "Using the Business Process Editor."

Installing the Identity Manager IDE

This section provides instructions for installing and configuring the Identity Manager IDE application.

NOTE Identity Manager projects were changed significantly in the version 7.1 release and the version 7.1 `.nbm` requires JDK 1.5 and NetBeans 5.5. Consequently, you must upgrade any projects created with the version 7.0 `.nbm`. See "Upgrading Version 7.0 Projects" on page 4 for instructions.

Before You Begin

To run the Identity Manager IDE, you must have

- NetBeans 5.5 running under JDK 1.5 installed on your local system. You can download NetBeans 5.5 from the following location:

  http://www.netbeans.org/

- Configurator-level access to Identity Manager.

NOTE The default Identity Manager project provides a `mail.jar`, an `activation.jar`, and a `jms.jar` in the `custom/WEB-INF/lib` directory. These `jar` files are required by the bundled Tomcat (which is shipped with NetBeans by default).

If you are going to use a different target application server, you might need to remove these `jar` files. Refer to the application server product documentation for specific information about its `jar` requirements.

Upgrading Version 7.0 Projects

To upgrade from the 7.0 `.nbm` to the 7.1 `.nbm`, you must perform the following steps:

1. Upgrade your JDK installation to JDK 1.5.
2. Upgrade you current NetBeans installation to NetBeans 5.5.
3. Install the Identity Manager 7.1 `.nbm` in your NetBeans 5.5 installation.
4. Create a new Identity Manager project.

   You can now create a *regular* project or a *remote* project depending on your needs. See "What is a Project?" on page 26 for more information about these project types.

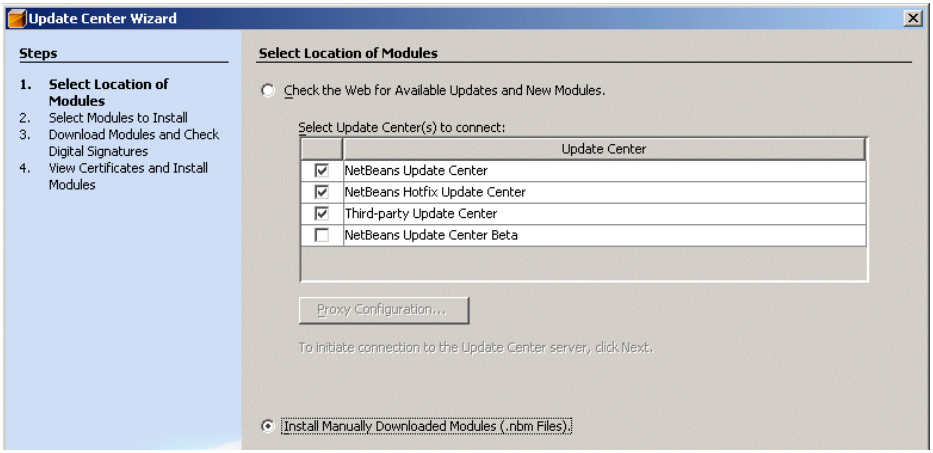
5. Copy the contents of your old project's `objects` directory to the `src/objects` directory of your new project.

Installing the Module

Identity Manager IDE is packaged as a module plug-in that you must register with NetBeans. To install and register the module, perform the following steps:

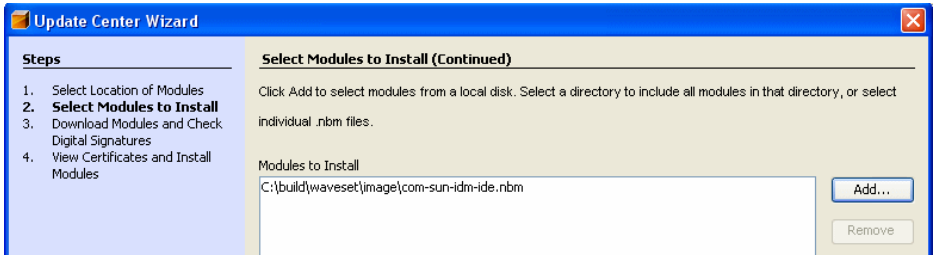
1. Start NetBeans 5.5.
2. From the NetBeans menu bar, select Tools > Update Center.
3. When the Update Center Wizard displays (Figure 1-1), select Install Manually Downloaded Modules, and then click Next.

Figure 1-1 Selecting the Module Location



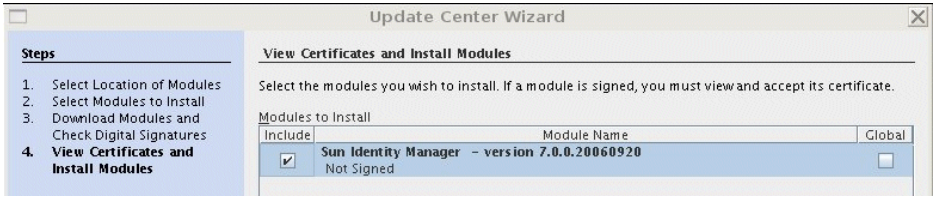
4. In the Select Modules to Install panel (Figure 1-2), click Add to locate and select the Identity Manager IDE NetBeans plug-in file (`com-sun-idm-ide.nbm`), which file is provided in Identity Manager's `image` directory.

Figure 1-2 Specifying the Module to Install



5. Click Next and the Select Modules to Install panel displays. Click Next again to download the module.
6. When the Identity Manager IDE License Agreement displays, click Accept.
7. The Download Modules panel displays to show you the status of the module(s) being downloaded. Click Next.
8. The View Certificates and Install Modules panel displays (Figure 1-3), containing a list of the module(s) you downloaded. Enable the check box beside the module name.

Figure 1-3 Select the Module to View Its Certificate



9. A pop-up displays, asking if you really want to install an unsigned module.
   - Click Yes to continue installing the module.
   - Click No if you do not want to continue the installation.
10. Click Finish.

NetBeans installs the Identity Manager IDE module.

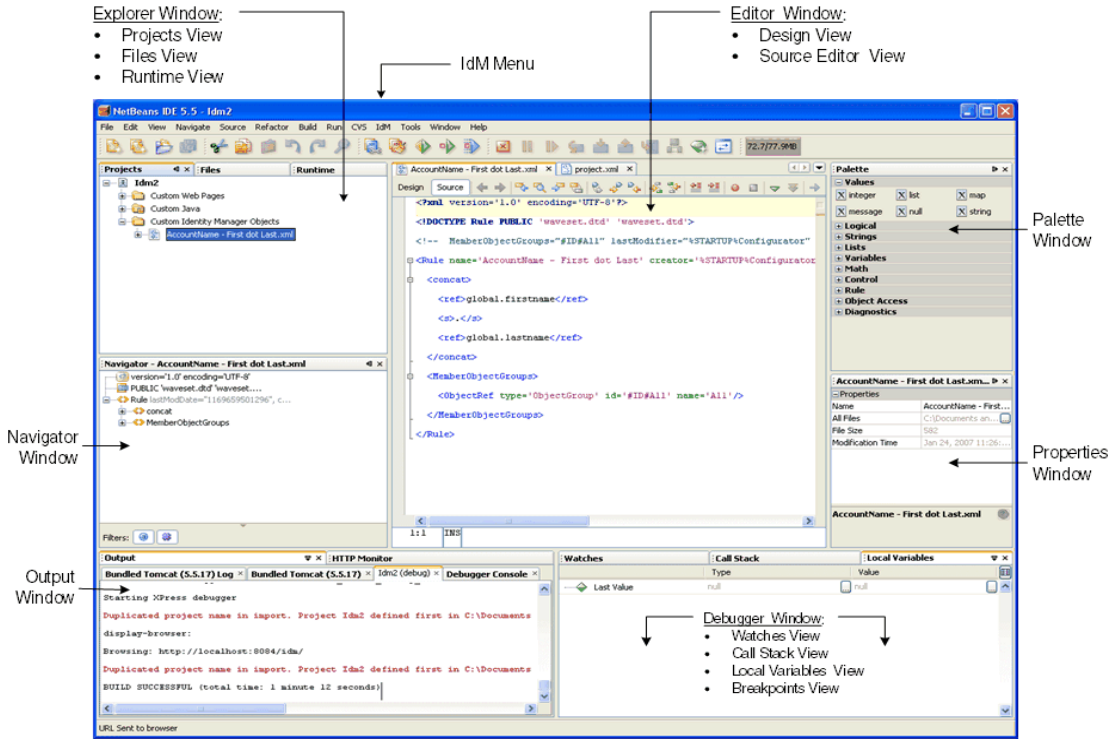
You can now

- Create Identity Manager projects within NetBeans
- Download objects from the Identity Manager installation
- Create new XML object files to upload

Working with the Identity Manager IDE Interface

The Identity Manager IDE interface (shown in Figure 1-4) consists of an IdM menu (located on the NetBeans' top-level menu bar) and the following windows:

Figure 1-4 Identity Manager IDE User Interface



NOTE

- Not all of these windows display when you first open the Identity Manager IDE, but they will open automatically (or can be opened) as you use different features in the application.
- You will encounter features (such as menu options) that are part of the NetBeans user interface. These features are not discussed in this publication. Consult the NetBeans online help or product documentation for information about these features.

This section introduces the IdM menu and each of the Identity Manager IDE windows. The information is organized as follows:

IdM Menu

The IdM menu becomes available from NetBeans' top-level menu bar after you have installed and configured the Identity Manager IDE application.

You can use the IdM menu to select actions that are appropriate to whatever nodes are currently selected in the NetBeans Explorer. For example,

- If you select a folder, you can upload or download objects.
- If you select an Identity Manager XML file, you can upload or reload objects.

Table 1-1 describes all of the IdM menu options.

TIP

- Most IdM menu options are also available on a pop-up menu that displays when you right-click objects in the Project tree.
- Additional information about using these IdM options is provided in the "Managing the Embedded Repository," "Working with Repository Objects," and "Working with the Identity Manager IDE Debugger" sections.

Table 1-1 IdM Menu Options

Select	To Perform the Following Action
Repository > Explore	Open the Explore Repository window so you can browse the contents of the Identity Manager IDE repository. You can select one or more objects and then • Click the Download button to download the objects to your local file system. • Click the Delete button to remove selected objects from the Repository.
Repository > Open Object	Open an object in the Editor window. **Note**: You can only open objects on your local file system. If you try to open an object that has not been downloaded from the repository yet, Identity Manager IDE automatically downloads the object to your local file system, and then opens it in the Editor.
Repository > Checkout View	Check out a particular view (such as a user view) from the repository for editing.
Repository > Upload Objects	Upload objects from your local file system to the repository. See page 48 for more information.
Repository > Diff Objects	Compare all objects in a directory on your local file system with their counterparts in the repository. Identity Manager IDE diffs all of the objects in the directory recursively down through the file system and then provides a side-by-side comparison of the Local and Remote XML code with syntax highlighting and line numbers.
Repository > Manage Embedded Repository **Note**: This menu option is not available for Identity Manager Project (Remote) or if you specify your own repository.	Open the Manage Embedded Repository dialog so you can modify the following repository settings for the current project: • **Repository Location**: Specify a different repository location. • **Initialize Repository**: If enabled, Identity Manager IDE removes existing data files and imports the `init.xml` file from the `idm.war`. Do not enable this option if you simply want to change the location of your repository to one that is already initialized. • **Automatically Publish Identity Manager Objects**: If enabled, Identity Manager IDE automatically uploads all Identity Manager objects in the project to the repository every time you run or debug the project.
Repository > Reload Object	Replace (overwrite) an object in the current project with a new version of that object from the repository.
Run LH command	Execute `lh` commands (such as `lh console` or `lh setup`) from within the Identity Manager IDE.

Table 1-1 IdM Menu Options *(Continued)*

Select	To Perform the Following Action
Set Identity Manager Instance	Open the Set Identity Manager Instance dialog, where you can control for which Identity Manager instance the following actions apply in the context of a given project: Debug Project Repository > Explore Repository > Open Project Checkout View Upload Object(s) Diff Object(s) Reload Object Test Form Test Rule For more detailed information, see "Setting the Identity Manager Instance" on page 36.
Clear Credentials Cache	Cause Identity Manager IDE to forget passwords it remembered when users enabled the Remember Password option.
Options	• Remove all auto-generated repository IDs that conform to a specified expression — without removing any hard-coded, predefined IDs — before Identity Manager IDE downloads objects from the repository. This feature is useful if you want to move objects from one repository to another. • Specify Diff Options by enabling or disabling the Exclude lastModDate, Exclude Ids, or Apply pattern substitution options. See "Using the Diff Options" on page 58 for more information about diffing objects.
Test Form	Test forms in a browser.
Test Rule	Verify modifications made to standalone and library rules.

Explorer Window

The Explorer window, located in the upper right corner of the Identity Manager IDE provides three windows:

- Projects Window
- Files Window
- Runtime Window

Projects Window

The Projects window displays in the Explorer by default when you open the Identity Manager IDE, and it provides a vertical, logical view of all open projects.

Expand the project node for a hierarchical view of the XML objects in the project, which includes:

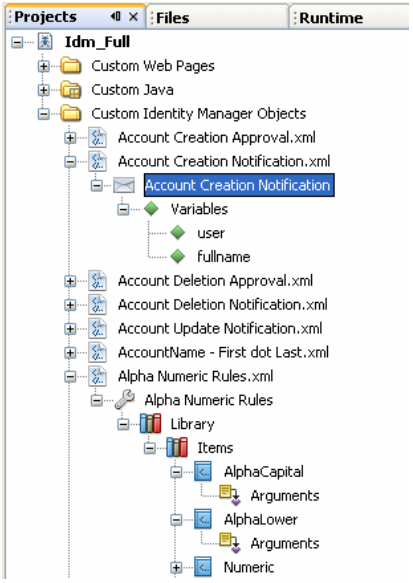
- Configuration Objects
- Email Templates
- Forms
- Generic Objects
- Libraries
- Rules
- Workflow Processes
- Workflow Subprocesses

Most elements that make up an XML object are also represented by nodes in the project tree.

NOTE By default, the object nodes are presented in the same order within the Project tree as they occur in the XML object. However, you can change their order in the tree (and in the XML object) using the Change Order, Insert Before, Insert After, Move Up, or Move Down pop-up menu options. See Table 1-2 for a description of these options.

Figure 1-5 shows various top-level XML files and some variable, library, rule, and argument nodes.

Figure 1-5 Projects Window



When you right-click on any node in this window, a pop-up menu displays with options that enable you to perform different tasks related to that node.

NOTE Some of the menu options are provided by the NetBeans application, and are not discussed in this publication. Consult the NetBeans online help or product documentation for information about these options.

Table 1-2 provides an overview of the pop-up menu options that pertain to Identity Manager IDE.

Table 1-2 Identity Manager IDE Pop-Up Menu Options

Node	Menu Options	Description
Project >	Debug Project	Select to debug the project.
	Close Project	Select to close the project.
	Repository >	Select to explore the repository; open, upload, or diff objects; check out views; or manage the embedded repository.
Object Node >	Add	Select to create a new object for this project.
	CVS	Select to manage CVS versioned-controlled files from within Identity Manager IDE.
	Repository	Select to download objects from the repository to your local file system, upload objects from your local file system to the repository, and to checkout a view.
	Find	Select to search your project for text, object names, object types, and dates. You can search all projects in the Project window, or select and search a specific location.
	Cut, Copy, Paste	Select these options to cut and paste, or copy and paste, objects.
	Delete	Select to delete selected objects from the project.
	Refactor	Select to modify the object's code structure and update the rest of your code to reflect the modifications.
	Tools	Select to create JUnit tests, add to Favorites, manage Internationalization, or to apply a Diff patch.
	Properties	Select to view and edit the selected object's properties (such as TaskDefinition Properties, Configuration Properties, and Repository Properties).
	Add *element*	Select to add actions, arguments, activities, forms, object references, processes, properties, results, returns, rules, transitions, or variables to the object. **Note**: The object node type determines which elements can be added. For example, you can add arguments, forms, results, returns, and variables to an Action object.
	Change Order	Select to change the order of multiple objects at once. A Change Order dialog displays so you can move multiple objects up or down to change their position in the Project tree and in the XML object.
	Insert Before	Select to move a selected object node above a specified object node in the Projects tree.
	Insert After	Select to move a selected object node below specified object node in the Projects tree.
	Move Up	Select to move a selected object node up the Projects tree.
	Move Down	Select to move a selected object node down the Projects tree.

Files Window

Select the Files window and expand the nodes for a directory-based view of your projects, including files and folders that do not display in the Projects Window.

You can open and edit your project configuration files, such as the project's build script and the properties files.

NOTE When you create a new project, Identity Manager IDE automatically creates a `README.txt` file.

To see a detailed overview of your project's file system (or if you want to modify the project structure to work with a different Configuration Build Environment (CBE)), double-click `README.txt` in the Files window and the information displays in the Editor window.

Runtime Window

Select the Runtime window for read-only access to the Identity Manager catalog (`waveset.dtd`) under the DTD and XML schema catalogs.

Editor Window

The Editor window provides a toolbar and a display area that enable you to work with objects in Design view or Source Editor view.

- **Design view**: Use this view to work with
  - An Expression Builder to work with Rule objects. (See "Design View For Rules" on page 15 for additional information.)
  - A graphical representation of Workflow Process and Workflow Subprocess objects. (See "Design View For Workflows" on page 16 for additional information.)

NOTE When you are working with rules and workflows, Design view is the default view.

- **Source Editor view**: Use this view to work with the object's XML source. (See "Source Editor View" on page 17 for additional information.)

The remaining buttons on the toolbar change, depending which view is selected.

NOTE You can drag items from the Palette window into the Editor window. For example:

- You can drag new Workflow services, approvals, users and Workflow tasks into the Editor for workflows.
- You can drag new fields into the Editor for forms.

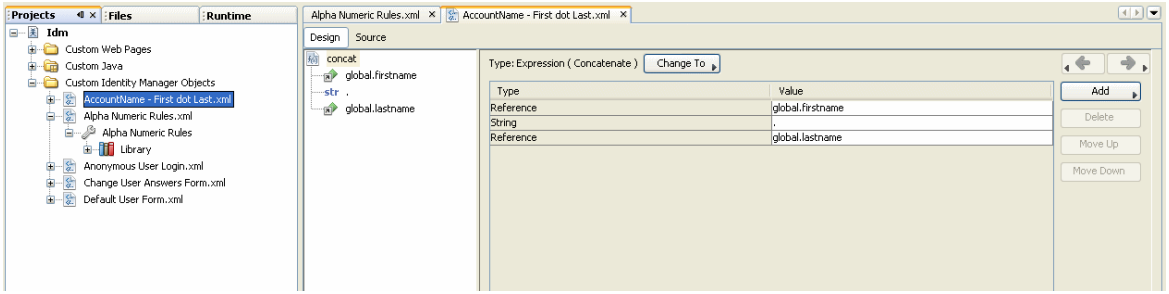
Dragging items into Design view, updates the XML source and the node tree in both the Projects and Files views. (See "Palette Window" on page 20 for more information.)

Design View For Rules

This section describes the Design view editors for rules and workflows.

The Design view editor for rules provides an Expression Builder (Figure 1-6) that makes it easier for you to see the logical structure of a rule and to modify the rule's properties.

Figure 1-6 Example Design View for Rules



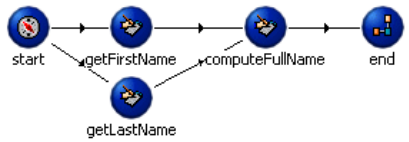
To open a rule in Design view, double-click the rule in the Projects window or select the rule node, and then click the Design button.

Instructions for using the Expression Builder are provided in "Using the Expression Builder Dialog" on page 53.

Design View For Workflows

The Design view editor for workflows provides a graphical diagram of a workflow (Figure 1-7), where each node represents a specific process activity and the lines represent transitions.

Figure 1-7 Example Workflow Design View



To work with the Design view for workflows, double-click a workflow in the Projects window or select the workflow node and then click the Design button.

In Design view, you can use the toolbar to perform the following tasks:

Table 1-3 Design View Toolbar Buttons

Click this Button	To Perform this Task
Add Activity:	Click to add activities to the Workflow.
Remove Activity:	Click to remove one or more Activities from the workflow. (To delete multiple activities, press the Ctrl key while selecting nodes.)
Layout:	Click to layout the workflow. Resets and organizes activities if they are arranged in a chaotic fashion.
Validate XML:	Click to validate the workflow. The validation information displays in the Output window, and typically contains hyperlinks that take you to a line in the Source Editor View so you can view or fix the XML.

In addition to these buttons, you can use the Workflow Selector menu (also provided on this toolbar) to select and work on a workflow or workflow subprocess in Design view.

When you edit a workflow in Design view, the Palette window displays items loaded from the workflow library. You can drag these items from the Palette and drop them into Design view to create an activity node in the diagram (which also updates the XML source and the node tree in the Project and Files views). See "Palette Window" on page 20 for more information.

Source Editor View

The Source Editor is a full-featured text editor that displays a selected object's unfiltered XML. In addition, the Source Editor is integrated with the Explorer Projects window and the Debugger.

You can open the Source Editor view by:

- Double-clicking an editable object in Projects window
- Creating a new, editable object from the available Identity Manager IDE templates (which opens the Source Editor automatically)
- Clicking the Source button at the top of the Editor window (only available for Rules and Workflows)

Figure 1-8 Example Source Editor View

Table 1-4 describes the toolbar buttons (from left to right) provided for the Source Editor view. The table also provides keyboard shortcuts for these tasks.

Table 1-4 Source Editor View Toolbar Buttons (and Shortcuts)

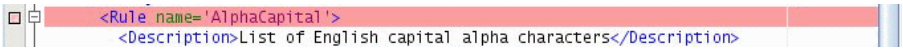
Use this Button	To Perform this Task
	**Back** (Alt+K) and **Forward** (Alt+L): Click these buttons to move back to the last edit, or forward to the next edit, you made in the Source Editor window.
	Use these buttons to move the cursor's insertion point as follows: • **Find Previous Occurrence** (Shift+F3): Click this button to locate and move the insertion point to text that you previously searched for. • **Find Selection** (Ctrl+F3): Click this button to locate the item where your cursor is currently inserted. • **Find Next Occurrence** (F3): Click this button to locate and move the insertion point to the next occurrence of text that you are searching for.
	Use these buttons to toggle the features on or off as follows: • **Toggle Highlight Search** (Alt+Shift+H): Click this button to turn highlighting search text on and off. • **Toggle Bookmark** (Ctrl+F2): Highlight a line and then click this button to insert a bookmark on, or remove a bookmark from, the current line.
	**Next Bookmark** (F2) and **Previous Bookmark** (Shift+F2): Click these buttons to locate the next bookmark or previous bookmark in the XML source.
	**Next Matching Word** (Ctrl+L) and **Previous Matching Word** (Ctrl+K): Select a word in the XML source, and then click these buttons to locate the next or previous occurrence of that word.
	**Shift Line Left** (Ctrl+D) and **Shift Line Right** (Ctrl+T): Click these buttons to reduce or increase the indentation of a selected line by one tab stop.
	**Start Macro Recording** (Ctrl+J S) and **Stop Macro Recording** (Ctrl+J E): Click these buttons to start or stop recording a macro containing keystrokes and cursor movements.
	Use these buttons to check or validate your XML source as follows: • **Check XML** (Alt+F9): Click this button to check whether you object's XML is correct and well-formed. The Output window identifies errors in your XML source, and typically provides a hyperlink to the line in the Source Editor so you can correct the problem. • **Validate XML** (Alt+Shift+F9): Click this button to validate the object's XML against the dtd. Validation information displays in the Output window, and typically contains hyperlinks to the line in the Source Editor View so you can view or fix the XML.

Setting Breakpoints

You can set breakpoints (distinct stopping points in process execution) in the Source Editor by clicking in the left margin immediately adjacent to an XML tag. For example, if you click in the margin next to the `<WFProcess . . .>` tag, you can set a breakpoint at the start of the workflow.

Figure 1-9 shows an example breakpoint.

Figure 1-9 Example Breakpoint



Dragging from the Palette

When you select objects for editing, the Palette window (see page 20) automatically displays (typically in the upper right side of the Identity Manager IDE) with items from the workflow library and XPRESS categories. You can drag items from the Palette and drop them into a line in the Source Editor to create XML text at that point in the XML source.

Inserting Invokes

When you right-click in an open XML file in the Source Editor, a pop-up menu displays that provides the following Insert options:

- **Insert Instance Invoke**: Calls an instance method on a Java object that is provided in the first argument. If the invoke does not have a classname attribute defined then it is a instance invoke type.
- **Insert Static Invoke**: Calls a static Java method. If the invoke has a classname attribute defined then it is a static invoke type.

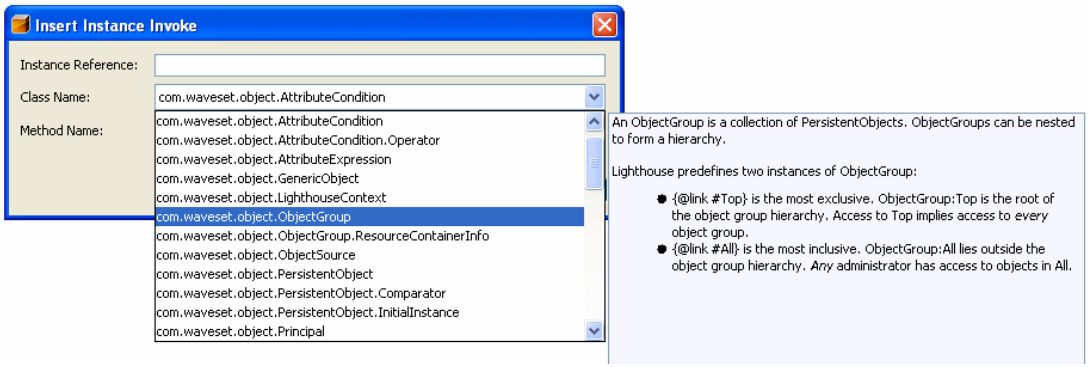
Selecting either option opens a Insert Invoke dialog where you specify

- Instance Reference (*for Instance Invokes only*)
- Class Name
- Method Name (Your class selection determines which methods are available.)

TIP	Clicking the drop-down arrows on the Class Name and Method Name menus opens JavaDoc for the classes or methods, respectively. As you run the cursor over names in the list, the related JavaDoc displays in a pop-up alongside the dialog.

Figure 1-10 shows an example Insert Invoke dialog with the JavaDoc display.

Figure 1-10 Example Insert Instance Invoke Dialog with JavaDoc



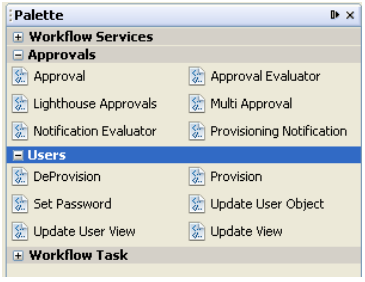
When you click OK, and Identity Manager IDE automatically generates the XML text at that point in the XML source.

NOTE When working with Workflow processes or subprocesses, you can add *Action* expressions to perform simple calculations or call out to Java classes or JavaScript to perform complex operations.

Palette Window

The Palette window (such as Figure 1-11) enables you to "drag-and-drop" elements into Email Template, Form, Library, Workflow Process, or Workflow Subprocess objects displayed in the Editor windows — without having to type XML.

Figure 1-11 Example Palette Window for Delete User Object



For example, you can drag fields into Form objects, rules into Library objects, or workflow services, approvals, users, and workflow tasks into Workflow Process and Workflow Subprocess objects.

When you double-click objects in the Project tree, the Palette window displays (typically in the upper right side of the Identity Manager IDE) and provides access to different elements — depending on the editing view, as follows:

- **Design view** provides a graphical representation of items loaded from the workflow library. Dragging items from the Palette into Design view creates an activity node in the diagram.
- **Source Editor view** displays the XML source of items loaded from the workflow library and XPRESS categories. Dragging items from the Palette into the Source Editor creates XML text wherever you drop the item in the XML source.

Dragging items into either Editor window updates both the XML source and the node tree in the Projects and Files views.

NOTE After adding elements to an object, you can change the order of those elements in the Project tree and in the XML object using a variety of pop-up menu options. See Table 1-2 for a description of the Change Order, Insert Before, Insert After, Move Up, and Move Down options.

Properties Window

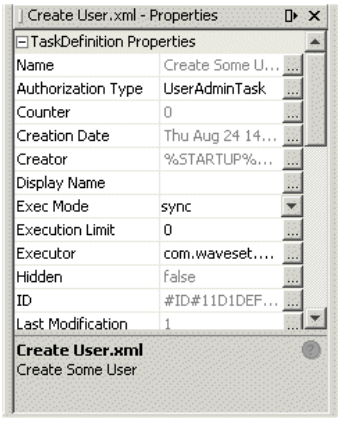
The Identity Manager IDE Properties window consists of a properties sheet for XML elements associated with Email Template, Form, Library, Rule, Workflow Process, and Workflow Subprocess objects. You can use this properties sheet to view and edit a selected object's properties; including the object name, file sizes, modification times, result information, and so forth.

Use one of the following methods to open the Properties window:

- Select Window > Properties from the main menu bar
- Right-click a node in the Projects window and select Properties from the menu.

When the Properties window is open, you can double-click an object node to update the contents of the window.

Figure 1-12 Example Properties Window



NOTE Instructions for using properties sheets to modify an object's XML elements are provided in "Editing Objects" on page 50.

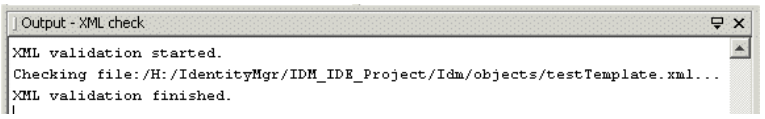
TIP As an alternative to the using Properties window, right-click object nodes in the tree and select Properties from the pop-up menu. A Properties dialog displays with the same object property sheet you would see in the Properties window.

Output Window

The Output window displays messages from the Identity Manager IDE when it finds errors during debugging. These messages usually include hyperlinks to the lines in your source code that caused the errors.

The Output window opens automatically when you run the Debugger, or you can open this window by selecting Window > Output from the main menu bar.

Figure 1-13 Example Output Window



Debugger Windows

The Identity Manager IDE Debugger is similar to both the Java Debugger and the Debugger provided with the BPE — you can set line, global, view, or form cycle breakpoints in the code, start the Debugger, and have the code stop in the XML.

In addition, the Identity Manager IDE Debugger is integrated with a rule tester and form tester, and it enables you to add a watch to show the value of a particular variable. (These features are discussed later in this chapter.)

The Debugger provides the following windows to help you troubleshoot your XML source:

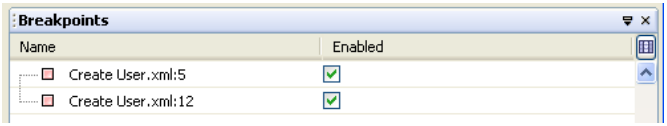
- Breakpoints Window
- Call Stack Window
- Local Variables Window
- Watches Window

Breakpoints Window

Breakpoints window lists all of the breakpoints you have set for the current project, including a short description of the breakpoint and a boolean flag that indicates whether the breakpoint is currently enabled or disabled.

Select Window > Debugging > Breakpoints from the main menu bar to open the Breakpoints window.

Figure 1-14 Example Breakpoints Window



You can enable or disable a breakpoint by changing its Enabled property in the Breakpoints window.

Call Stack Window

The Identity Manager IDE automatically launches the Call Stack window when you run the Debugger.

When the Debugger reaches a breakpoint, the *execution stack* (a series of calls that the thread performed to get to the breakpoint) displays in the Call Stack window. In XPRESS, these calls are XPRESS operations such as `BLOCK` or `CASE`.

If additional functions appear in the call chain, these functions are listed in order. This list is called a *stack trace*, and it displays the structure of the execution stack at this point in the program's life.

NOTE If you close the Call Stack window (or it is unavailable for another reason) you can re-open the window by selecting Window > Debugging> Call Stack from the main menu bar.

Local Variables Window

The Local Variables window lists all variables that are in scope at the current point of execution when the Debugger stops at a breakpoint. Variables represent values of the following types:

- Simple (such as strings)
- Complex, which contain other values (such as lists and maps)

  You can expand complex types to show their contained elements.

The Local Variables window remains blank when debugging is inactive or when execution is not stopped at a breakpoint.

If the current element is an XPRESS end tag, the result of the last evaluation displays in the Local Variables window when you stop the Debugger. This evaluation value also applies to other tags for which last value makes sense. For example, as Identity Manager evaluates `<Argument>` tags to workflow subprocesses, the argument value shows in the Local Variables window.

NOTE

- The Local Variables window does not display variables while XPRESS execution is in progress.
- The Last Value entry remains blank when debugging is inactive.

Watches Window

The Debugger allows you to use *watches*, which enable you to track changes made to variable or expression values when you run the Debugger. A watch can be any valid XPRESS expression.

When you specify a watch, the Watches window lists all of the variables, expressions, types, and values that are being watched when you debug the project. This list may also contain hyperlinks to take you to the object type in the XML source.

The Identity Manager IDE automatically launches the Watches window when you run the Debugger (or you can select Window > Debugging > Watches from the main menu bar).

Using Keyboard Shortcuts

Through NetBeans, Identity Manager IDE enables you to use different keyboard combinations to run commands and navigate in the Identity Manager IDE, including:

- Menu shortcuts help you navigate, activate, and select menu options from the menu bar.
- Source Editor shortcuts help you execute commands and actions in the Editor. Some of these shortcuts are common to all file types, while others are only available when editing certain file types.
- Window shortcuts help you navigate, activate, and select operations in the Identity Manager IDE windows.
- Help shortcuts help you navigate the Identity Manager IDE Online Help System.

For a detailed description of available keyboard shortcuts and instructions for using them, refer to the online help.

NOTE To assign shortcuts, select Tools > Options, and when the Options dialog displays, select Keymap.

Working with Identity Manager IDE Projects

To use Identity Manager IDE effectively, you must also understand some basic concepts about the Identity Manager IDE *project*. This section provides the following information:

What is a Project?

A *project* is a collection of source files (JSPs, Java, and Identity Manager objects), build scripts, and so forth that control all of the customizations for a deployment.

Projects are tied to a specific repository. You can associate more than one project with a repository, but only one repository per project.

NOTE Identity Manager IDE projects allow you to interact directly with CVS. Refer to your NetBeans documentation for information.

You can create one of the following project types for Identity Manager IDE:

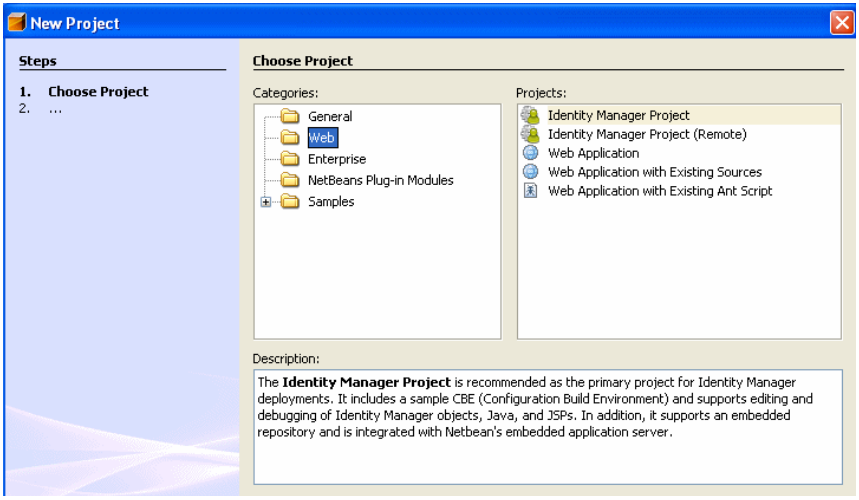
- **Identity Manager Project**: Fully featured project provided as the primary development environment for deployers. Highlights include:
  - Full build environment
  - Ability to manage custom Java code and JSPs
  - Integrated with NetBeans embedded application server to allow local deployment of the Identity Manager `war` file
  - Integrated Java, JSP, workflow, form, and XPRESS debugging
  - Embedded database repository
- **Identity Manager Project (Remote)**: Less-featured project for making small modifications and debugging on an external server. This project has all the editing functionality of the full-blown Identity Manager project, but lacks the build environment and ability to launch the war.

Creating a Project

To create a new project, use the following steps:

1. Select File > New Project.
2. When the New Project wizard displays (Figure 1-15), select Web from the Categories list to indicate what type of project you are creating.

Figure 1-15 New Project Wizard: Specify a Category



3. Select one of the following sample Identity Manager projects from the Projects list:
   - **Identity Manager Project**: Select this project for a fully featured development environment, which includes:
     - A sample Configuration Build Environment (CBE)
     - The ability to manage custom Java code and JSPs
     - Integration with the NetBeans embedded application server, which allows you to deploy the Identity Manager `war` locally
     - Integrated Java, JSP, workflow, form, and XPRESS debugging
     - An embedded database repository

- **Identity Manager Project (Remote)**: Select this project to make small modifications and to perform debugging on an external server.

  This project type has all of the editing functions as the full Identity Manager project, but it does not provide the build environment or give you the ability to launch the Identity Manager `.war` file (`idm.war`).

4. When you are finished, click Next.

   - If you selected the full project, continue to the section titled, "Creating the Identity Manager Project."
   - If you selected the Remote project, continue to the section titled, "Creating the Identity Manager Project (Remote)."

The Identity Manager IDE now works with different versions of the server.

Each Identity Manager IDE project is tied to a specific Identity Manager version, and the Identity Manager IDE requires a compatibility bundle (a zip file that provides Identity Manager `jar` files and some XML registries containing version-specific information for each supported Identity Manager version.

- For **Identity Manager Projects**: The compatibility bundle is included in the `idm.war` file, and Identity Manager IDE automatically accesses this file during project setup.
- For **Identity Manager Projects (Remote)**: Because you do not specify a `war` file location for remote projects, you must provide the compatibility bundle's location, which is:

  `<Identity Manager install root>/sample/ide-bundle.zip`

NOTE Currently, you cannot manage multiple Identity Manager versions from the same project. If you want to perform simple debugging or make simple adjustments to a server for a different version, create a separate remote project (as described on page 33) for that server.

You also cannot use the Identity Manager IDE to upgrade your existing projects to a newer version. You must perform this task manually, as follows:

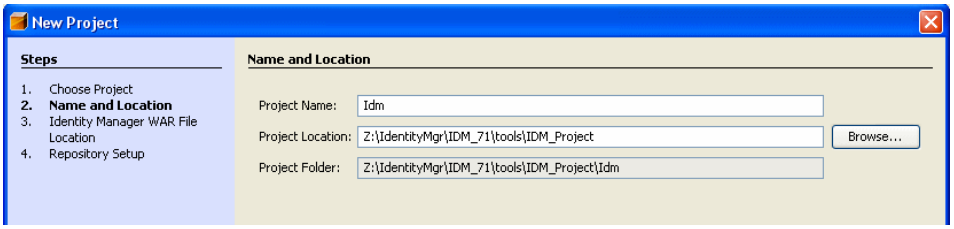
1. Shutdown NetBeans.
2. Manually upgrade all of your objects, custom JSPs, and the contents of your project's idm-staging directory.
3. Replace the existing `nbproject/ide-bundle.zip` with that from the version to which you are upgrading.

Creating the Identity Manager Project

To finish creating the full project, perform the following steps:

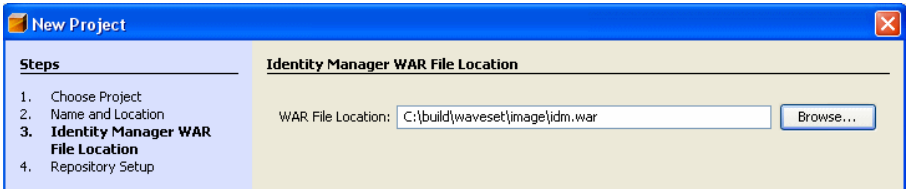
1. The Name and Location panel displays with a default project name, location, and folder in which to store the Identity Manager project.
   - If the default location is satisfactory, click Next.
   - If you want to specify a different location, replace the default information and then click Next.

Figure 1-16 New Project Wizard: Specifying Where to Store the Project



2. When the Identity Manager WAR File Location screen displays, specify the path to the Identity Manager `idm.war` file, and then click Next.

Figure 1-17 New Project Wizard: Specifying the WAR File Location



3. When the Repository Setup screen displays, (Figure 1-18) select one of the following options for configuring a repository:

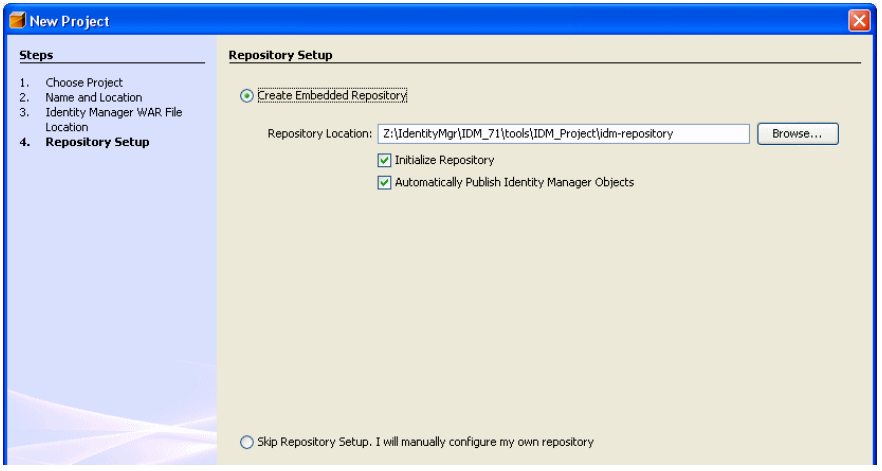
   - If you want Identity Manager IDE to create an embedded repository, enable the Create Embedded Repository option, and continue with Step 4.

   NOTE When you select Create Embedded Repository, Identity Manager IDE creates an embedded repository for the default *sandbox* CBE target.

   If you specify additional targets, you must also select Manage Embedded Repository or you must run the `lh setup` or `setRepo` commands for the additional targets.

   - If you want to reuse an existing repository from another project, enable the Skip Repository Setup option, and continue with the instructions provided in "Configuring a Repository Manually" on page 32.

Figure 1-18 New Project Wizard: Repository Setup



4. Specify a directory for the embedded repository data files.

5. Enable Initialize Repository to remove existing data files and import the `init.xml` file from the `idm.war`.

   NOTE Be sure to *disable* this option if you want to reuse a repository from another project.

6. Enable Automatically Publish Identity Manager Objects to automatically publish all Identity Manager objects in the project to the repository every time you Run or Debug the project.

   NOTE This option is not available for Identity Manager Project (Remote) or if you specify your own repository.

7. Click Finish.

   A Creating Project dialog displays while Identity Manager IDE sets up the repository. This process will take some time as Identity Manager IDE removes data files and imports the `init.xml` file.

   The setup is complete when the dialog closes and you see a `BUILD SUCCESSFUL` message display in the Identity Manager IDE Output window.

You can select the Files tab in the Explorer window and expand the top-level node to view the files associated with your new project.

NOTE When you create a new project, Identity Manager IDE automatically creates a `README.txt` file.

To see a detailed overview of your project's file system (or if you want to modify the project structure to work with a different CBE), double-click `README.txt` in the Files window and the information displays in the Editor window.

Configuring a Repository Manually

You can configure your project to use any of the repository types supported by Identity Manager.

NOTE Manually configuring a repository impacts the repository for the currently selected CBE target. For more information, see "Setting the Identity Manager Instance" on page 36.

Because configuration instructions vary for each repository type — the following example is provided to illustrate the process:

To configure a MySQL repository, perform the following steps:

1. Copy `mysqljdbc.jar` into your project's `custom/WEB-INF/lib` directory.
2. In the Identity Manager IDE Project window, right-click on the *project name* and select Run LH Command from the pop-up menu.
3. When the Enter LH Command to Run dialog displays, type **`setup`** into the LH Command field, and then click OK.

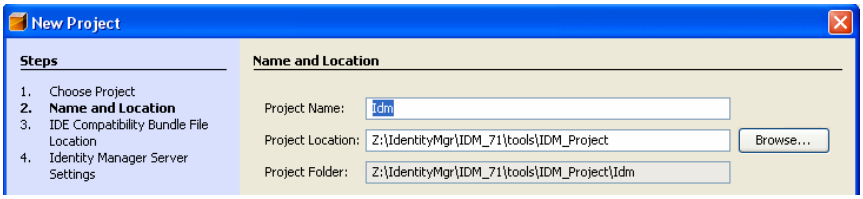
   The Input field displays below the Output window and the Sun Setup Wizard displays with some introductory information.
4. After reading the information, click Next.
5. When the Locate the Repository screen displays, provide the following information and then click Next:
   a. Select the repository type from the menu.
   b. Verify the URL, JDBC Driver, and connection information and modify if necessary.
6. The Setup Demo? screen displays, asking if you want to continue setting up a demonstration environment. Click No, I will configure Identity Manager myself, and then click Next.
7. When the Save Configuration screen displays, click Execute to import the `init.xml` file and save the Identity Manager configuration.
8. When the import is finished, click Done.

Creating the Identity Manager Project (Remote)

To finish creating a remote project, perform the following steps:

1. The Name and Location panel displays with a default project name, location, and folder in which to store the Identity Manager project.
   - If the default location is satisfactory, click Next.
   - If you want to specify a different location, replace the default information and then click Next.

Figure 1-19 New Project Wizard: Specifying Where to Store the Project



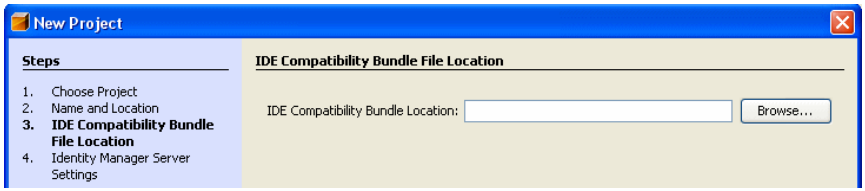
The Identity Manager IDE Compatibility Bundle File Location screen displays.

NOTE Because Identity Manager IDE can work with different versions of the server, and every Identity Manager IDE project is tied to a specific Identity Manager version, you must specify an IDE compatibility bundle (a zip file that provides Identity Manager `jar` files and some XML registries that contain version-specific information) for each supported Identity Manager version.

2. Specify the location of the IDE Compatibility Bundle for the version of the server to which you are connecting, and then click Next.

   For remote projects, the IDE Compatibility Bundle File is located in `<Identity Manager install root>/sample/ide-bundle.zip`.

Figure 1-20 New Project Wizard: Specifying the IDE Compatibility Bundle File Location



3. When the Identity Manager Server Settings screen displays (Figure 1-21), enter the following information to define how you are going to connect to the Identity Manager instance for development, and then click Finish.

NOTE

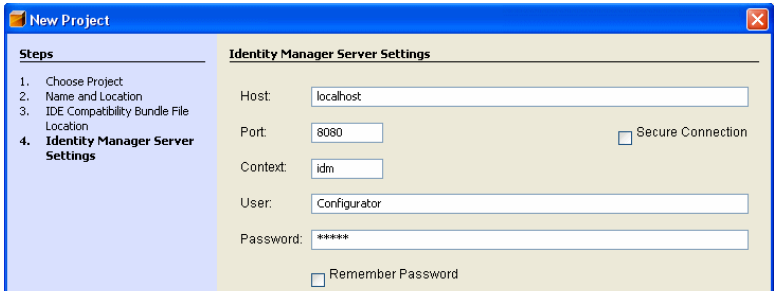
- Identity Manager IDE supports remote (SOAP) connections.
- When you set up a remote connection, Identity Manager IDE uses this connection to upload or download objects.

- **Host**: Enter the name of the host where your project is running.
- **Port**: Enter the number of the TCP port on which the directory server is listening.

  Enable the Secure Connection box if you want Identity Manager IDE to use SSL when opening the connection.
- **Context Path**: Enter the context root for your Identity Manager installation (for example, `/idm`)
- **User**: Enter your administrator user name.
- **Password**: Enter your administrator password.

  Enable the Remember Password box if you want Identity Manager IDE to remember and automatically enter your password for future sessions.

Figure 1-21 New Project Wizard: Specifying the Server Settings



4. Click Finish.
   - If the connection to your remote server is successful, the project node displays in the Identity Manager IDE Projects window (upper left corner).
   - If the connection fails, an error message displays in the wizard window to provide information about the failure. Correct the problem and click Finish again.

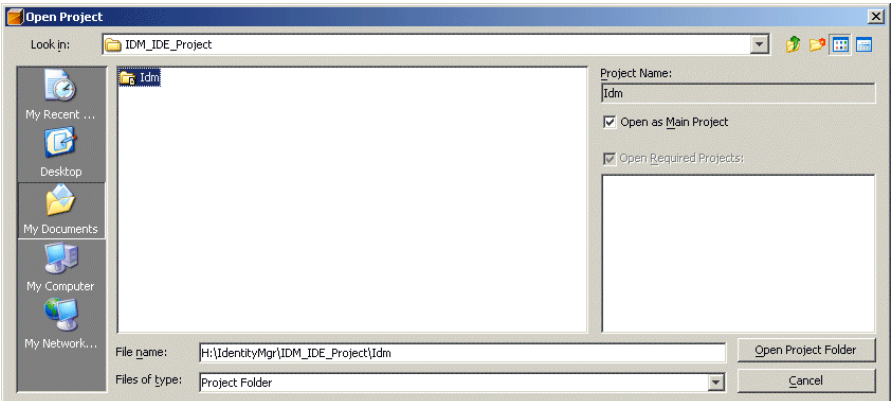
TIP	• Remember that you are not connecting to the embedded server. If you experience a connection failure, verify that your remote server is running. • You can also test connections in Identity Manager IDE when you edit an existing project's properties.

Choosing an Existing Project

To open an existing project, use the following steps:

1. Select File > Open Project.
2. When the Open Project dialog displays (Figure 1-22), browse to the project folder you want to use and click Open Project Folder.

Figure 1-22 Open Project Dialog



If you already have one or more projects open, the Open as Main Project check box (upper right corner) becomes active.

3. Enable this box if you want to specify this project as the main project.
4. Click Open Project Folder and the selected project is added to the Explorer window.

Setting the Identity Manager Instance

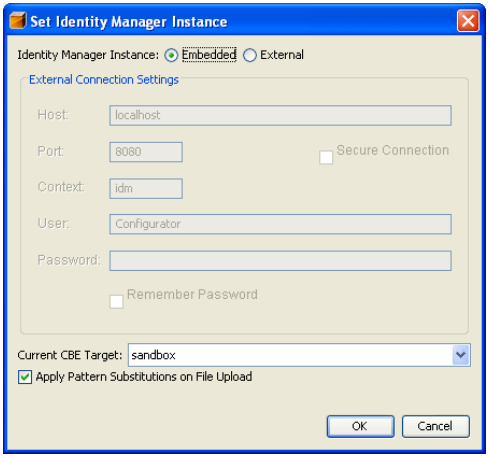
You use the Set Identity Manager Instance feature to control for which Identity Manager instance the following actions apply, in the context of a given project:

- Debug Project
- Repository > Explore
- Repository > Open Project
- Checkout View
- Upload Object(s)
- Diff Object(s)
- Reload Object
- Test Form
- Test Rule

Use one of the following methods to open the Set Identity Manager Instance dialog (Figure 1-23):

- From the NetBeans main menu bar, select IdM > Set Identity Manager Instance.
- From the Projects window, right-click the project node and select Set Identity Manager Instance from the pop-up menu.

Figure 1-23 Set Identity Manager Instance Dialog



Reviewing Project Connection Settings

To view a project's external connection settings, enable one of the following Identity Manager Instance options:

- **Embedded** (*not available for remote projects*): Applies all of the following actions to the NetBeans embedded application server:
    - Checkout View
    - Debug Project
    - Diff Object(s)
    - Reload Object
    - Repository > Explore
    - Repository > Open Project
    - Test Form
    - Test Rule
    - Upload Object(s)
- **External**: Applies all of the preceding actions to the server specified by the external connection settings.

Setting the Current CBE Target

To specify the current CBE target for all build actions (Build Project, Run Project, and Debug Project),

1. Select a target from the Current CBE Target menu.

NOTE The selected target also controls which repository is used or set by specifying `%%CUSTOM_SERVER_REPOSITORY%%` in each of the `*-target.properties` files.

For example, if you set the CBE target to `dev` and select IdM > Repository > Manage Embedded Repository (or you select IdM > Run LH Command and enter `setRepo` or `setup`) Identity Manager IDE looks in `dev-target.properties` to get `%%CUSTOM_SERVER_REPOSITORY%%`, which is `nbproject/private/dev-ServerRepository.xml`. Identity Manager IDE then sets `nbproject/private/dev-ServerRepository.xml` to the specified repository.

- When you perform a Build Project action, the generated `war` file contains a `ServerRepository.xml` equal to `nbproject/private/dev-ServerRepository.xml`.
- When you perform a Run Project or Debug Project action, Identity Manager IDE launches the `war` file in NetBean's embedded application server with a server repository equal to `nbproject/private/dev-ServerRepository.xml`.

---

2. Enable the Apply Pattern Substitutions on File Upload box if you want the Identity Manager IDE to apply the pattern substitutions defined in `<`*target-name*`>.properties` whenever you upload one or more files to the repository. (Where *target-name* is the target selected by Current CBE Target.)

To verify how this feature works with multiple targets, create two targets with two different repositories, as follows:

1. Create a new (full-featured) Identity Manager project and, when the Repository Setup screen displays, select Create Embedded Repository.
2. Click Finish to create the repository and to set the current target to *sandbox*.

3. Launch the application server.

   From the Projects tab, right-click the project node and select Run Project from the menu.

4. When the Identity Manager browser opens, log in and create a user named **`sandbox-user`**.

   After creating the account, select the List Accounts tab and you should see `sandbox-user` listed.

5. Shutdown the application server.

   From the Identity Manager IDE Runtime tab, expand the Servers node, right-click Bundled Tomcat, and then select Stop from the pop-up menu.

6. Once the server has stopped, return to Identity Manager IDE Projects tab, right-click the project node, and select Set Identity Manager Instance.

7. When the Set Identity Manager Instance dialog displays, change the Current CBE Target to *dev*, and then click OK.

8. When you are prompted to clean the project, select Yes.

9. Next, create a repository for the dev target.

   a. Right-click the project node and select Repository > Manage Embedded Repository from the menu.

   b. Specify a different location for the dev target than the one used for the first repository.

   c. Enable the Initialize Repository option, and then click OK.

10. When you see the `BUILD SUCCESSFUL` message in the Output window, right-click the project node and select Run Project.

    Notice that the NetBeans log states it is `Preparing image for environment target: dev`.

11. Log in to Identity Manager and create a new user named **`dev-user`**.

    Now, when you select the List Accounts tab, the new `dev-user` should be listed, but the `sandbox-user` is not.

12. Switch back to the `sandbox` target.

    a. Shut down the server.

    b. Right-click the project node, select Set Identity Manager Instance > Current Target > sandbox.

c. When you are prompted to clean the project, select Yes.

d. Right-click the project node and select Run from the menu.

13. Log into Identity Manager, select List Accounts, and notice that `sandbox-user` is now available.

Managing the Embedded Repository

You can modify the following repository settings for the current project by selecting IdM > Repository > Manage Embedded Repository:

NOTE Managing the Embedded Repository will impact the repository for the currently selected CBE target. For more information, see "Setting the Identity Manager Instance" on page 36.

- **Repository Location**: Specify a different repository location.
- **Initialize Repository**: If enabled, Identity Manager IDE removes existing data files and imports the `init.xml` file from the `idm.war`.

  Disable this option if you want to reuse a repository from an existing project.
- **Automatically Publish Identity Manager Objects**: If enabled, Identity Manager IDE automatically uploads all Identity Manager objects in the project to the repository every time you run or debug the project.

  NOTE This option is not available for Identity Manager Project (Remote) or if you specify your own repository.

Working with Repository Objects

Identity Manager IDE enables you to work with repository objects on your local file system, rather than working directly in the repository (as with the BPE).

This section provides the following information:

Supported Object Types

You can work with the following object types in the Identity Manager IDE:

- **Configuration Object**: A persistent object that contains forms and workflow processes.
- **Email Template**: Templates used to send notification of various changes and actions to users and administrators. You might use an email template, for example, to notify an approver of a pending request, or to notify a user that his password has been reset.
- **Form**: An object associated with a Web page that contains rules about how a browser should display user view attributes on that page. Forms can incorporate business logic, and often are used to manipulate view data before it is presented to the user.
- **Generic Object**: An object typically used to represent views, such as simple collections of name/value pairs. Generic objects have an `<Extension>` of type `<Object>`.
- **Library**: An object used to organize closely related objects, typically rules, into a single object in the repository. Organizing objects into libraries makes it easier for workflow and form designers to identity useful objects.

- **MetaView**: In Identity Manager, a meta view is a unified view of all resources, providing a common data model in which to view a set of resources and how attributes on these resources flow from one to another.
- **Rule**: Objects in the Identity Manager repository that contain functions written in the XPRESS, XML Object, or JavaScript languages. Within Identity Manager, rules store frequently used logic or static variables for reuse in forms, workflows, and roles.
- **Workflow Process**: A workflow is a logical, repeatable process during which documents, information, or tasks are passed from one participant to another. Identity Manager workflows comprise multiple processes that control user accounts, such as account creation, update, enabling, disabling, and deletion.
- **Workflow Subprocess**: An object used to create a workflow subprocess to include in a workflow.

Checking-Out Views

The Identity Manager IDE allows you to check out a particular view (such as a user view) from the repository for editing.

NOTE In general, do not use the Checkout View feature to update a user. This feature is provided only to help you understand view behavior. To update users in Identity Manager, use the Identity Manager Web interface.

To check out a view,

1. Right-click Custom Identity Manager Objects folder in the Project window and select Repository > Checkout View from the pop-up menu.
2. When the Checkout View dialog displays, select a View type from the menu, enter View name, and then click OK.

Identity Manager IDE opens the view contents in tree format.

After editing, right-click on the view to check it back into the repository — just like any other XML object.

Getting Objects from the Repository

One advantage to using the Identity Manager IDE, is that it enables you to download and modify objects outside of the repository. You can also, if necessary, reload an object from the repository to replace one you modified on your local file system.

This section describes the different methods for getting objects from the repository, including:

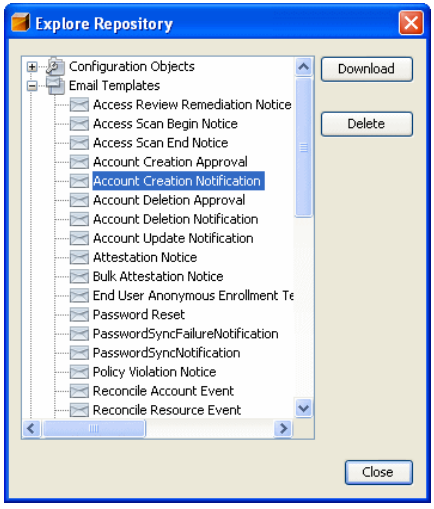
- Downloading Objects
- Opening Objects
- Reloading Objects

Downloading Objects

To download an object from the repository to your local files system for the first time, use the following steps:

1. Right-click the Custom Identity Manager Objects node and select Repository > Explore to open the Explore Repository dialog displays (Figure 1-24).

Figure 1-24 Explore Repository Dialog



2. Expand the nodes to browse the objects in the Repository, and select the object you want to download from the tree.

TIP	You can download multiple objects from the repository by pressing the Ctrl key as you select the objects from the list.

3. Click Download.

Identity Manager IDE adds the selected object node, and children that represent individual elements in the object, to the Custom Identity Manager Objects tree in the Explorer. Double-click the child nodes to view their XML elements in the Editor and to view their properties sheet in the Properties window.

Opening Objects

If you already know the name of an object you want to open on your local file system, use the Open Objects option to quickly locate and download a single object from within a larger set of objects (such as the repository).

To use the Open Objects option,

1. Select IdM > Repository > Open Object from the IdM menu (or right-click the Custom Identity Manager Objects node and select Open Object).

   When the Open dialog displays.

2. Use the options on this dialog to specify your object search criteria:

   a. **Object Type**: Select object types for which to search.

NOTE	*Common Configuration Object Types* is the default selection, which enables you to search for any of the configuration objects loaded within the system by default (Configuration, Email Template, Form, Generic Object, Library, Rule, Workflow Process, and Workflow Subprocess). Generally, you do not have to change the Object Type unless you want to search for a different object type or your search results exceed the maximum number of results.

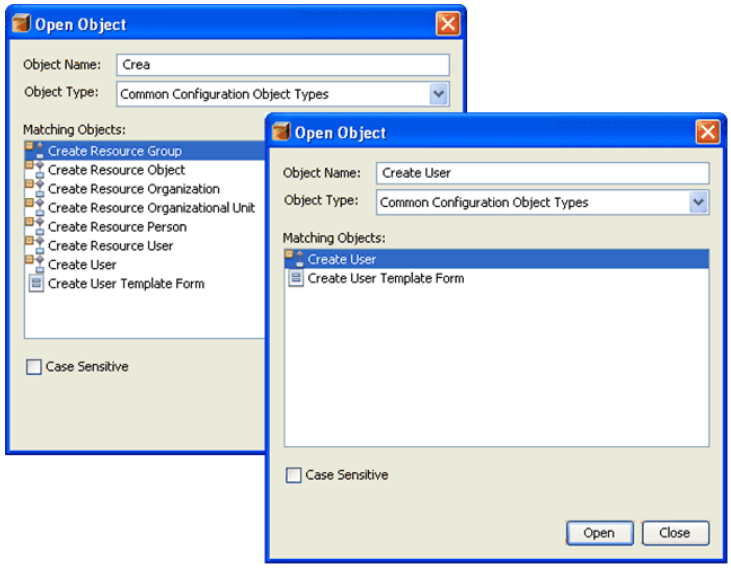
   b. **Object Name**: Type all or part of the object name into the text field.

      You can specify an asterisk (*) as the object name to access all objects of a particular type in the repository.

As you begin typing the object name, a list of matching items displays in the Matching Objects field. Each new letter narrows the displayed results. For example, if you type **`Creat`** (for Create User), a large number of results display (for example see Figure 1-25).

As you type more of the entry, such as **`Create`**, the results narrow because the criteria becomes more specific.

Figure 1-25 Open Object Dialog



c. **Case Sensitive**: Enable this box to constrain the search to object names that match the case used in the Object Name field.

3. Select the object you want to open in the Matching Objects list, and then click Open (or press the Enter key) to open that object in the Identity Manager IDE.

NOTE

- The Matching Objects list can return a maximum of 1000 objects. If your search results exceed this number, a message displays to suggest you refine the search criteria.
- If you try to open an object that has not been downloaded from the repository yet, Identity Manager IDE automatically downloads the object to your local file system, and then opens it in the Editor.

Opening Object References

You can right-click nodes in the Project tree to open objects to which certain references refer (such as ObjectRefs, FormRefs, FieldRefs and Workflow Subprocesses). When you right-click an object node representing the object in the tree, you can select Open *object name (*where *object name* is Reference, Form, Field, or External Process respectively) from the pop-up menu.

- If the object to which the reference refers exists on your local file system, the Identity Manager IDE opens the source code from the local file system in the Source Editor.
- If source code for the object does not yet exist on your local file system, the Identity Manager IDE retrieves the object from the repository and then opens the source code in the Source Editor.

For example, to add a Default User Library object reference to the Tabbed User Form:

1. Download the Tabbed User Form and expand the nodes until you see the Includes node.
2. Right-click the Includes node and select Add an Object Reference.

   The object reference does not yet exist on your local file system, so the Add Object Reference dialog displays.
3. You can browse and select an object or type the object name directly into the Name field. Select the Default User Library object, and then click OK to add the new object under the Includes node in the tree.
4. Now, when you right-click the new Default User Library object reference node and select Open Reference, the Identity Manager IDE downloads the Default User Library object reference form and opens the XML source in the Source Editor.
5. Close the object reference form in the Source Editor.
6. Right-click the object reference node again and open the Default User Library reference.

   The Default User Library form opens in the Source Editor, but it is not downloaded because it now exists on your local file system.

Reloading Objects

To replace (overwrite) an object in the current project with a new version of that object from the repository, use the following steps:

1. Right-click the object node and when the pop-up menu displays, select Repository > Reload Objects.

   TIP You can download multiple objects from the repository by pressing the Ctrl key as you select the objects from the list.

2. An Overwrite Files? dialog displays to let you know that the object already exists on your local file system, and asking you to confirm that you want a fresh copy from the repository.
   - Click Yes or Yes for All to continue.
   - Click No or No for All if you do not want to proceed.
3. When you are ready, click the Reload button.

The selected object nodes, and children that represent individual elements in the object, will be displayed in the Explorer. You can double-click child nodes to go to their XML elements and view them in the property sheet.

Removing Object IDs

To facilitate moving objects from one repository to another, you can configure Identity Manager IDE to remove all auto-generated repository IDs before it downloads objects from the repository.

NOTE In general, you should not have to modify the regular expression pattern; the default should be sufficient. This feature is provided in case the default pattern is broken.

Identity Manager IDE locates and removes all object IDs and objectRef IDs that conform to a specified expression — without removing any hard-coded, predefined IDs.

To configure this feature,

1. Select IdM > Options to open the Repository Options dialog.
2. Enter a regular expression in the ID Removal Pattern field, and then enable the Use ID Removal box.

NOTE

- The Use ID Removal box is enabled by default.
- Identity Manager IDE stores the regular expression and displays the most recently used expression in the ID Removal Pattern field by default, which allows you to quickly reuse expressions you used previously.

Uploading Objects to the Repository

To upload new or modified objects from your local file system back to the repository, use the following steps:

1. Select the one or more object nodes in the Projects window.

   TIP You can select multiple objects to upload by pressing the Ctrl key as you select the objects from the list.

2. Right-click a node and select Repository > Upload objects from the pop-menu.

   Identity Manager IDE immediately uploads the selected objects.

Uploading Objects Automatically

When you create a new project, you can enable a Automatically Publish Identity Manager Objects option and Identity Manager IDE will automatically upload all Identity Manager objects to the repository whenever you run or debug the project.

See "Creating the Identity Manager Project" on page 29 for instructions.

You can enable this option after project creation from the Manage Embedded Repository dialog (IdM > Repository > Manage Embedded Repository).

NOTE This option is not available for Identity Manager Project (Remote) or if you specify your own repository.

Using CBE Pattern Substitution

Identity Manager IDE allows you to use a CBE (Configuration Build Environment) pattern substitution file when uploading objects to an embedded or external Identity Manager instance.

When you select IdM > Set Identity Manager Instance from the main menu bar, the Set Identity Manager Instance dialog displays. If you enable Apply Pattern Substitutions on File Upload, the pattern substitutions defined in *target-name*`.properties` (where *target-name* is the target selected by Current CBE Target) will be applied whenever you upload one or more objects.

For example, `sandbox-target.properties` is the default target and it defines `%%SMTP_HOST%%=mail.xyzcompany.com`. Users can replace `mail.xyzcompany.com` in `sandbox-target.properties` with the name of their SMTP server, and then use the `%%SMTP_HOST%%` variable in the Identity Manager object files as appropriate.

NOTE

- The Identity Manager Project provides a sample CBE and the `nbm` is aware of the CBE pattern substitution files. The `*-target.properties` files are also used by the CBE build and the `publishIDMObjects ant` target.
- You can move the `*-target.properties` files to a different directory, but you must update the `build.xml` and the `cbeConfigDir` attribute in `idm-project.xml` to account for this change.

Creating New Objects

To create a new object for your project, use the following steps:

1. Right-click an object node (or an object types node) in the Projects window and select New > File/File Folder.
2. In the New File wizard, select the Sun Identity Manager Objects category, and then select an object type to create. Click Next.
3. In the Name and Location screen, enter a file name and specify where to store the file.

NOTE The current project, project directory, and objects folder are selected by default, but you can specify a different location.

4. Click Finish to close the wizard.

Identity Manager IDE adds the new object to the tree as a child under the selected object type node. In addition, the new object's XML source is displayed in the Source Editor view.

NOTE Identity Manager IDE assumes that each XML source file contains exactly one object.

Consequently, if you use XML files containing multiple objects (including some files distributed with Identity Manager, such the `wfresource.xml` sample), the objects do not display as nodes in the Project window and they are not available in the Palette. In addition, the context menu does not include any Identity Manager IDE actions.

However, if you put these XML files into your project's Objects directory, Identity Manager IDE handles the files as plain text so you can view the files in the Files tree, and you can open and edit the files directly.

Editing Objects

This section explains how to edit the objects in your project. This information includes:

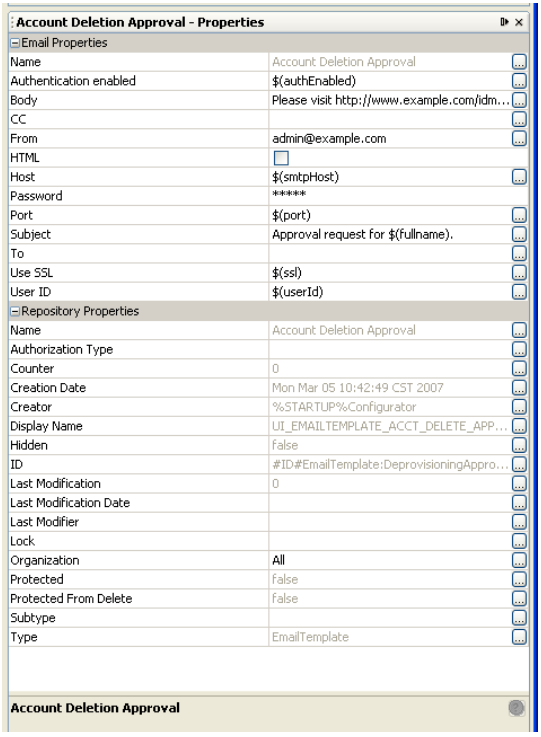
- Editing Object Properties
- Adding Elements to an Object

Editing Object Properties

You can modify properties for Email Template, Form, Library, and Workflow objects from the Identity Manager IDE Properties window.

For example, click the Account Deletion Approval object node and following properties sheet (Figure 1-26) displays in the Properties window:

Figure 1-26 Example Properties



TIP As an alternative to the using Properties window, right-click the object node and select Properties from the pop-up menu. A Properties dialog displays with the same object property sheet you would see in the Properties window.

Properties sheets contain some or all of the following features:

- **Text fields**: Allow you to edit simple or complex values, as follows.
  - **Simple value fields**: Type new string or integer values directly into the text field. After modifying or adding a new value,
    - Press the Enter key (or click in a different text field) to apply your changes.
    - Press the Esc key to cancel your changes.
  - **Complex value fields**: Click the existing value to open a Expression Builder dialog (see "Using the Expression Builder Dialog" on page 53) so you can edit the expression.

NOTE Greyed text indicates a read-only field.

- **Ellipsis (...) buttons**: Provide an alternative method for editing object values shown in a text field.
  - **Simple value fields**: Clicking the button opens a dialog so you can enter new string or integer values. After modifying or adding a new value,
    - Click OK to apply your changes and close the dialog.
    - Click Cancel to undo your changes and close the dialog.
  - **Complex value fields**: Clicking the button opens an Expression Builder dialog so you can edit the expression.

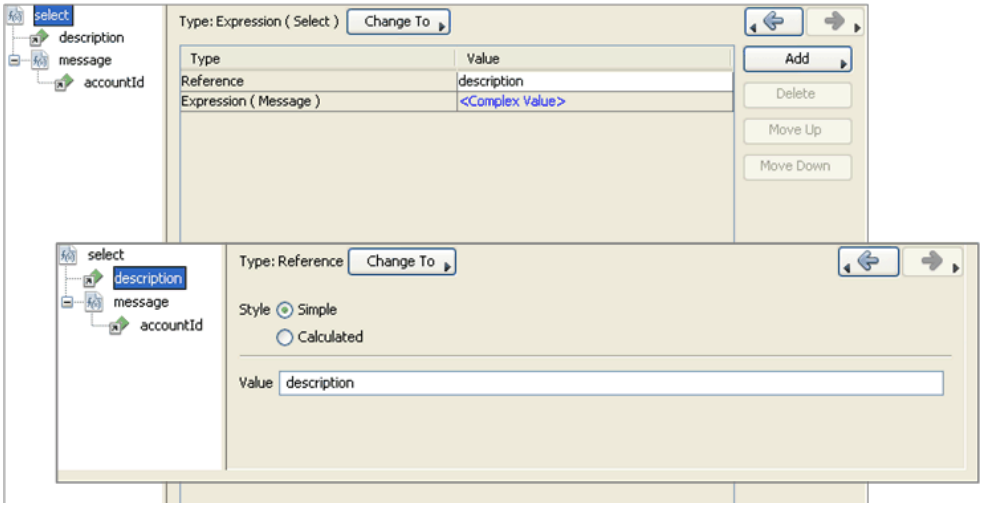
NOTE Greyed text indicates a read-only field.

- **Drop-down menus**: Select from a list of several, specific values or double-click the property name to rotate through the available values.
- **Check boxes**: Click to enable or disable the property.

Using the Expression Builder Dialog

When you are working in the Properties window, you can click a Complex value field or click the ellipses button to open an Expression Builder dialog (such as those shown in Figure 1-27) to edit an object's XML expressions.

Figure 1-27 Two Example Expression Builder Dialogs



Expression Builder dialogs consist of

- A tree view pane that provides a hierarchical view of the object's XML expressions.

  You can also right-click elements in this view to access options for editing expressions.

- An editing options pane that provides a set of editing options for the selected expression element.

- An information pane (located at the bottom of most editing options panes) that provides general and JavaDoc information related to the selected expression.

NOTE The Design view for rules also provides an Expression Builder (see Figure 1-6) to make it easier for you to see the logical structure of a rule and to modify the rule's properties.

Table 1-5 describes the different editing options:

NOTE For more information about the expression types mentioned in this table, see *Identity Manager Workflows, Forms, and Views*.

Table 1-5 Expression Builder Options

Editing Options	Description
• Add button • Add menu option (*only available by right-clicking elements in Tree view*.)	Add a new expression element by selecting one of the following expression types: • XPRESS Object > Select String, List, Map, or Integer. • Reference • Expression > Select Logical > *Logical type,* Strings > *String type,* Lists > *List type,* Variables > *Define a variable,* Math > *Math type,* Control > *Control type*. • Object Access > Select New, Invoke (Static or Instance), or Get. • Diagnostics > Select Script, Trace, or Print. • Rule • XML Object > Select String, List, Map, or Integer.
• Delete button • Delete menu option (*right-click elements in Tree view*.)	Delete the selected expression.
Move Up/Move Down buttons	Move elements up or down within the XML expression.

Wrap Object option (*only available by right-clicking elements in Tree view*.)	Wrap an expression element (such as a `string`) within another expression element (such as a `concat`).
	This option is useful if you forget to include a function call that needs arguments you already have defined. For example, if you define a reference to the `firstname` attribute and wrap that reference in a `concat`, you can then define a reference to the `secondname` attribute as an argument to the `concat` — resulting in the following:
	`<concat>`
	`<ref>firstname</ref>`
	`<ref>secondname</ref>`
	`</concat>`

Table 1-5 Expression Builder Options *(Continued)*

Editing Options	Description
• Change To button • Change To menu option (*right-click elements in Tree view*.)	Change an element's expression type to one of the following: • XPRESS Object > Select String, List, Map, or Integer. • Reference • Expression > Select Logical > *Logical type,* Strings > *String type,* Lists > *List type,* Variables > *Define a variable,* Math > *Math type,* Control > *Control type*. • Object Access > Select New, Invoke (Static or Instance), or Get. • Diagnostics > Select Script, Trace, or Print. • Rule • XML Object > Select String, List, Map, or Integer.
Arrow buttons 	Move between recently visited expression-editing panels (similar to a browser's back and forward buttons).
Type/Value table	View and edit the expression types currently used in the selected expression: • **Type column** (read-only): Lists the expression types. • **Value column** (editable): Lists the expression type values. To edit simple strings or integers, type a new value into the Value field. To edit a <Complex Value>, click the field to open the Expression Builder dialog for that expression.
Style buttons	Set the expression style to Simple or Calculated.
Value fields and <> Name fields	Replace the current simple string or integer.
Trace box	Output trace diagnostics while XPRESS is running inside IDM
Browse JavaDoc button	View JavaDoc information when you are creating invoke expressions. Click this button to open the JavaDoc dialog where you can select items from Class Names and Method Names menus. As you click (or mouse over) a menu item, a pop-up displays the related JavaDoc information.

To edit an expression,

- Select an element in tree view, and then use the different options on the editing options pane.

  Some editing options (such as the Change To button) are common to most expression elements, while others are only available when you are editing specific elements.

- Right-click an element in tree view, and then select an option from the pop-up menu.

Adding Elements to an Object

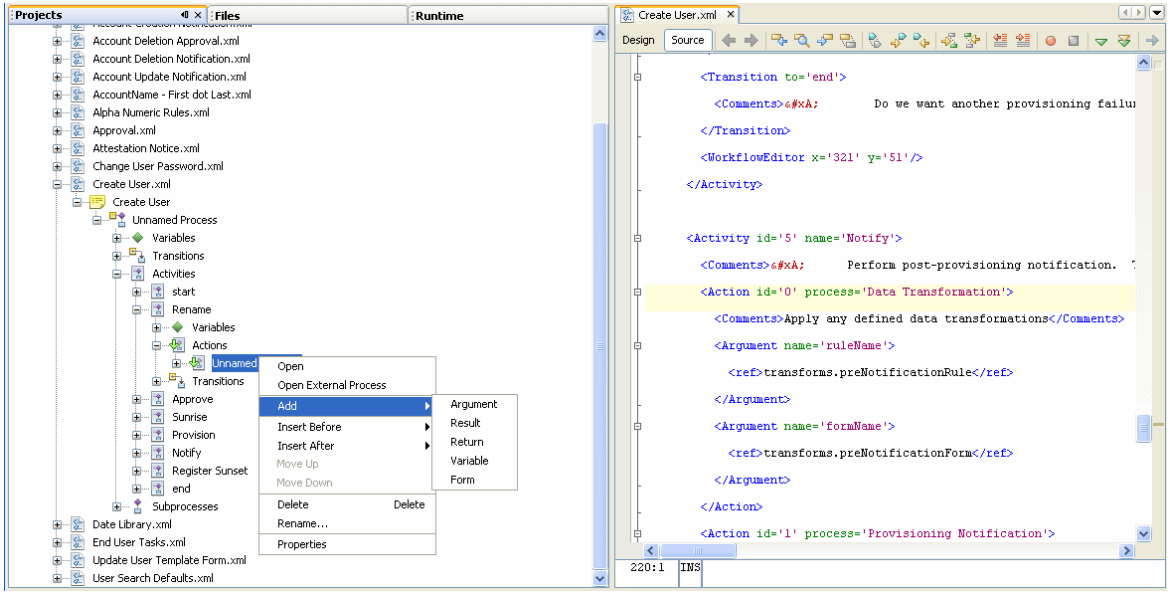
You can add new (child) elements to objects by editing the XML source directly or using the pop-up menu's Add option.

NOTE You can use the Properties Window to edit object properties, but you cannot use it to add elements.

To use the Add option,

1. Right-click the object node where you want to add the child element, and select Add *element_type* from the pop-up menu.

Figure 1-28 Adding Elements to an Action

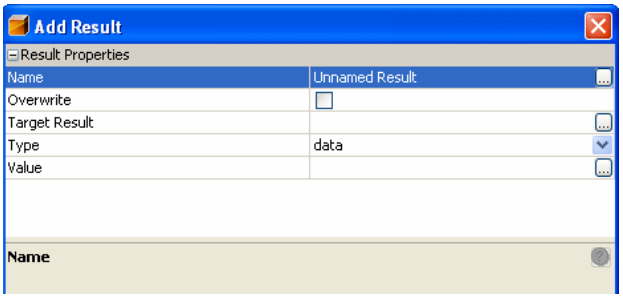


NOTE The object type determines which *element_types* can be added. For example, if you are working with Workflow processes or subprocesses, you can add *Actions* to perform simple calculations or call out to Java classes and JavaScript to perform complex operations. You can then add arguments, forms, results, returns, or variables to the Action object.

2. When the Add *element_type* dialog displays, use the property sheet to specify a name and other properties for the new element (see "Editing Object Properties" on page 51).

   For example, if you are adding a Result to an Action object, the Add Result dialog displays.

Figure 1-29 Add Result Dialog



3. When you click OK, the Identity Manager IDE adds the new element under the selected object node. In addition, the new element's XML source is displayed in the Source Editor view.

NOTE When adding an Include element to a form, you can either browse for and select the form name or you can type the name directly.

When typing a form name, you must use the following syntax or an error (`Error creating Object Reference`) will result:

`<FormType>:<SomeFormName>`

For example, to add the AIX User Form to the Default User Form

1. Download the `Default User Form.xml`, and expand the tree until you see the Includes node.

2. Right-click the Includes node and select Add Object Reference.

3. When the Add Object Reference dialog displays, you can either type the AIX User Form into the Name field or click Browse to locate and select the form.

4. If typing the name, you must enter **`UserForm:AIX User Form`**.

Deleting Objects

To remove objects from the project on your local file system,

1. Right-click on one or more objects in the Project window.
2. Select Delete from the pop-up menu.

   The objects are immediately deleted from your project.

To remove objects from the Identity Manager IDE repository:

1. Select IdM > Repository > Explore from the NetBeans menu bar.
2. When the Explore window displays, expand the object nodes to locate the objects you want to remove.
3. Select one or more objects, and then click the Delete button.

NOTE	You cannot use this Delete button to remove individual rules from a library. To delete one or more rules from a library, you must download the entire library to your local file system, delete the rules, and then upload them back to the repository.

   All deleted objects are immediately removed from the Repository.

Using the Diff Options

Identity Manager IDE allows you to compare (*diff*) directories or single objects on your local file system to their counterparts in the repository, and then view the differences side-by-side.

Setting Global Diff Options

Select IdM > Options to open the Repository Options dialog where you can specify the following global diffing options (which are all enabled by default):

- **Exclude lastModDate**: Controls whether Identity Manager IDE excludes `lastModDate` modifications when you diff objects.
- **Exclude ids**: Controls whether Identity Manager IDE excludes all auto-generated repository IDs when you diff objects.

- **Apply pattern substitution**: Controls whether Identity Manager IDE applies the `*-target.properties` file (a Configuration Build Environment (CBE) pattern substitution file) when you diff objects.

NOTE

- The Exclude lastModDate and Exclude ids options affect only those objects on the repository side.
- If you change any of the preceding options, Identity Manager IDE automatically updates the *active* window based on your selections.

  For example, if you click Options on the Identity Differences tab to open the Repository Options dialog and disable the Exclude ids option, the Identity Manager IDE automatically updates the information displayed on the Identity Differences tab, but *does not* update the Diff windows in the Editor.

Diffing Objects

After saving any modifications made to an object, you can diff the contents of a directory or of a single object, as described in the following sections:

- Diffing a Directory
- Diffing a Single Object

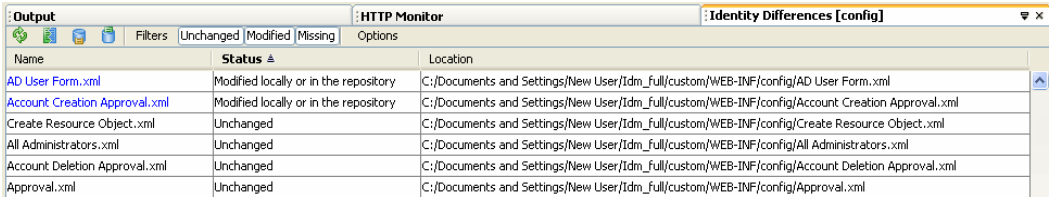
Diffing a Directory

To diff the contents of a directory,

- Select IdM > Repository > Diff Objects from the menu bar.
- Right-click the directory node (such as Custom Identity Manager Objects) and select Repository > Diff Objects.

Identity Manager IDE diffs all objects recursively down through the file system and then displays the results in the Identity Differences tab (Figure 1-30).

Figure 1-30 Identity Differences Tab



This tab consists of

- A table that lists the Name, Status, and Location of all diffed files.

  This table uses color-coded file names to indicate which files have been modified (blue) and which files are new and do not yet reside on the server (green).

  TIP Double-clicking a modified item in this table opens a Diff Output window so you can view the differences side-by-side.

- The icons and buttons described in Table 1-6.

Table 1-6 Identity Differences Tab: Diff Icons and Buttons

Click this Icon or Button:		To Perform this Task:
	**Refresh**	Retrieve any object changes from the local file system, re-diff those objects, and then redisplay the diff information on the Identity Differences tab. **Important**: You must click Refresh before using the Reload All, Upload All, or Diff All options to incorporate all of the newest changes for objects listed on the Identity Differences tab.
	**Diff All**	Diff all objects in a selected directory. When you diff the contents of a directory, the Identity Manager IDE also populates the Editor's drop-down menu with the names of any objects that have been modified locally or in the repository. Select an object from this list to view its diff windows in the Editor.
	**Reload All**	Replace all of the modified objects listed on the Identity Differences tab with their counterparts from the repository.
	**Upload All**	Upload all of the modified objects listed on the Identity Differences tab to the repository.
		Filter buttons: Select any combination of these buttons to control how results display in the Identity Differences table.
		• **Unchanged:** Filters all Unchanged results out of the list.
		• **Modified:** Filters all Modified locally or in the repository results out of the list.
		• **Missing:** Filters all Missing from Repository results out of the list.
	**Options**	Click to specify the global diffing options described in "Setting Global Diff Options" on page 58

NOTE You must click Refresh before reloading, uploading, or diffing all to retrieve the newest changes for objects listed on the Identity Differences tab. If you do not Refresh, the Identity Manager IDE uses the older version of the object displayed on the Identity Differences tab.

If you use any of icons or the Options button on the Identity Differences tab, the Identity Manager IDE automatically updates the information displayed in the Identity Differences tab — but does not update the Diff windows in the Editor.

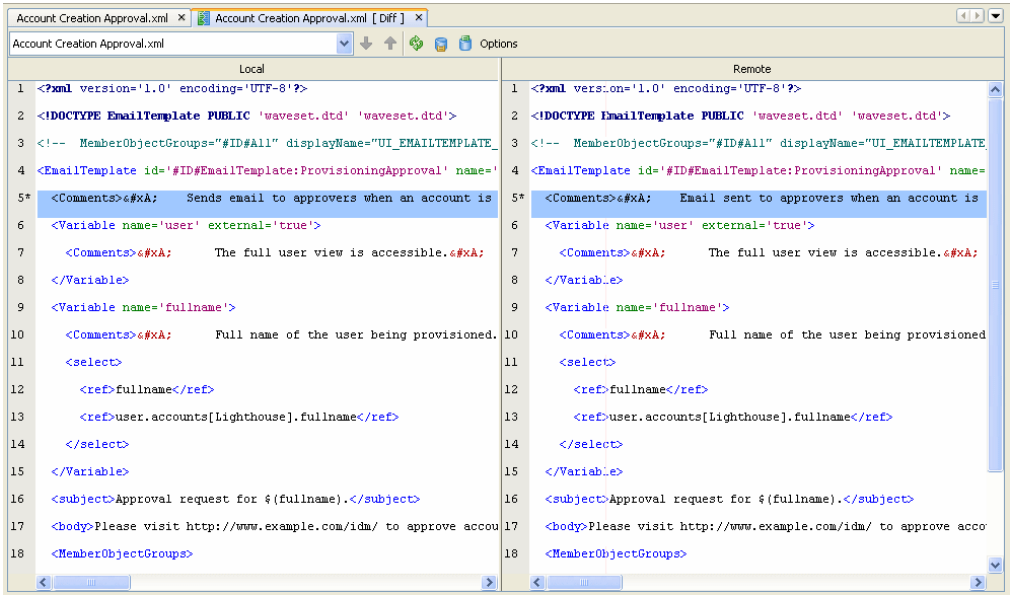
Diffing a Single Object

To diff the contents of a single object,

- Select IdM > Repository > Diff Object from the menu bar.
- Right-click the object and select Repository > Diff Object.

The Identity Manager IDE Editor window updates to display a Local view and Remote view of the object, as shown in Figure 1-31.

Figure 1-31 Example Side-By-Side Display



The Editor window also provides the icons and button described in Table 1-7:

Table 1-7 Editor Diff Icons and Button

Click this Icon or Button:		To Perform this Task:
	**Refresh**	Retrieve any object changes from the local file system, re-diff the object, and then redisplay the diff information in the Editor. **Important**: You must click Refresh before using the Replace local with repository or Upload local to repository options to include the newest changes made to that object.
	**Replace local with repository**	Replace a modified local object with its counterpart from the repository.
	**Upload local to repository**	Upload a modified local object to the repository.
	**Options**	Click to specify the global diffing options described in "Setting Global Diff Options" on page 58

NOTE If the Diffing windows are open, and you click any of the icons or the Options button from the Editor, Identity Manager IDE automatically updates the information displayed in the Diffing windows — but does not update the Identity Differences tab.

Working with XML

This section describes how to work with XML in the Source Editor window. The information is organized into the following sections:

- Editing XML
- Using Auto-Completion
- Identifying and Correcting Malformed XML
- Validating XML

Editing XML

To edit an object's XML,

1. Double-click the object node in the Project or Files window.
2. If the Source Editor view does not display by default, click the Source button.

When the object's unfiltered XML displays in the Source Editor; you can edit, check, and validate the XML as needed.

NOTE To edit an object that is not yet displayed in the Projects or Files window, select IdM > Repository > Open Objects. Specify all (or part) of the object name and the object type in the Open Objects dialog to locate that object on your local file system.

If you try to open an object that has not been downloaded from the repository, Identity Manager IDE automatically downloads the object to your local file system, and then opens it in the Editor.

The Identity Manager IDE provides several options for editing an object's XML:

- Type the information directly into the Source editor window.
- Right-click an object node in the Projects window and use the pop-up menus to access an Add dialog, where you can provide the necessary information.
- Use the Palette window, which displays items from the workflow library and XPRESS categories. You can drag items from the Palette and drop them into a line in the Source Editor to create XML text at that point in the XML source.
- Right-click in the XML file displayed in the Source Editor to insert Instance or Static Invoke statements. (See "Inserting Invokes" on page 19 for more information.)

Using Auto-Completion

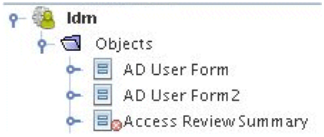
When you install the Identity Manager IDE, it registers the `waveset.dtd` definition file, which enables auto-completion for XML elements and attributes.

To use the auto-completion feature, begin typing an XML element or attribute, and then press Ctrl+Spacebar. NetBeans displays a list of the elements and attributes that can be used to automatically complete the element.

Identifying and Correcting Malformed XML

If you load a file into the Identity Manager IDE with malformed XML, or if you edit a file and the edits do not conform to the definition (DTD) file, a red error icon displays on the object's top-level parent in both the Project and File windows.

Figure 1-32 Error Icon



If the Properties window is open, an error message is displayed in the window's description area. Or, you can use the invalid file's tooltip information to view the error message by placing your mouse over the file's node.

When an object has malformed XML

- The object's child nodes do not display.
- The Add, Clone, and Upload commands, some context menus and buttons become disabled.
- You cannot use the Test Form or Test Rule features for Forms, Rule, or Library nodes.
- XML added directly to the XML editor is not lost, but a node for the added object will not display until you fix the XML.

If you right-click in the Source Editor window and select Validate XML from the pop-up menu, the results — including a link to the malformed XML source — display in the Output window. Click this link and Identity Manager IDE highlights the line just below the malformed XML in the Source Editor window so you can easily locate and correct the problem.

Validating XML

You can immediately validate a object's XML by right-clicking in the Source Editor and selecting Validate XML from the context menu.

Check the Output window for the results, which should be similar to the following message:

```
XML validation started.

Checking file:/H:/IdentityMgr/IDM_IDE_Project/Idm/objects/
System%20Configuration.xml...

XML validation finished.
```

Working with the Identity Manager IDE Debugger

The Identity Manager IDE provides a graphical Debugger that you can use to debug Identity Manager forms, rules, and workflows. You can use this Debugger to set breakpoints and watches, step through code, examine and modify variables, examine classes and the callstack, follow threads, and run multiple sessions.

This section describes how to use the Identity Manager IDE Debugger, and if you have previously used a code debugger for a procedural programming language, you should be familiar with the terms used here. The information is organized as follows:

- Starting the Debugger
- Setting Breakpoints
- Using Watches
- Stepping Through an Executing Process
- Debugging Forms
- Testing Rules
- Debugging Workflows
- Running the Debugger Outside a Test Environment
- Using the Identity Manager IDE Tutorial: Debugging Forms, Rules, and Workflows
- Stopping the Debugger

CAUTION The Identity Manager IDE Debugger is enabled by default, but you should always *disable* it when you are finished debugging to prevent someone from accidently connecting it to an application server in your production environment.

Instructions for disabling the Debugger, are provided in "Disabling the Debugger" on page 99.

NOTE You should be aware of the following information when using the Identity Manager IDE Debugger:

- *Do not use the Debugger in a production environment*.

  Setting a breakpoint is a global setting; consequently, the Identity Manager IDE suspends incoming request threads when it reaches that breakpoint.

- The user specified in the project's properties must have the Identity Manager Administrator capability.

  However, be aware that the Debugger can suspend threads that lock other users out of the system and display variables containing sensitive data from other users' sessions. Given the powerful repercussions of misuse, exercise caution when assigning the right to run the Debugger.

- You must assign a private copy of the application server to users running the Debugger.

  If multiple users are developing on the same application server and one user connects a Debugger to the server, the other users will hit breakpoints and be locked out.

- The Debugger does not support clusters.

- For more information about working with workflows, forms, and views refer to the *Sun Java™ System Identity Manager Workflows, Forms, and Views* publication.

Starting the Debugger

To start the Identity Manager IDE Debugger, select Run > Debug Main Project from the main menu bar.

When you start a debugging session, the Identity Manager IDE automatically opens a set of Debugger windows that display runtime information about your program. From these windows, you can

- Start and stop the debugging process
- Navigate through the process execution
- Set *breakpoints* (distinct stopping points in process execution)

Setting Breakpoints

You use *breakpoint* commands in the Debugger to halt the execution of an object before executing a specific line of code.

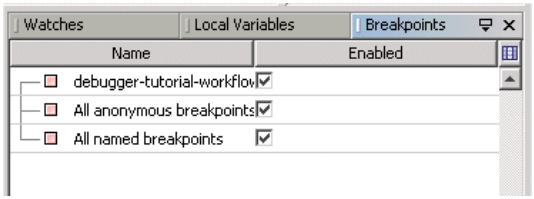
NOTE Be aware that setting a breakpoint is a *global* setting. The Debugger suspends incoming request threads when it reaches a designated breakpoint — and this action happens regardless of which user is making the request.

When you are using the Debugger, breakpoints apply regardless of where you launched the form or workflow. While most debuggers allow you to set breakpoints only on source locations; the Identity Manager IDE Debugger also permits you to set breakpoints at conceptual execution points, such as Refresh view. In this case, the Debugger suspends operation whenever a Refresh view operation occurs. You can then step-into the Refresh view and watch the underlying form processing in progress.

You can view a summary of all source breakpoints in the Breakpoints window (typically located in the lower-left corner of the Identity Manager IDE). You can also click on a breakpoint in the Breakpoints window to navigate to that breakpoint in the Source Editor.

When it stops at a breakpoint, the Debugger also displays variables (that are in scope) at the current execution point.

Figure 1-33 Breakpoints Window



The easiest way to set a breakpoint is to click in the left margin of the Source Editor, immediately adjacent to the tag where you want to add a breakpoint.

You can also select Run > New Breakpoint from the main menu bar. When the New Breakpoint dialog displays, select XPRESS from the Debugger menu. The content in the dialog changes to provide the following options:

- **Breakpoint Type menu**: Identity Manager is the only option that is available from this menu, and it is selected by default.
- **Global tab** (default): Select to enable one or both of the following options:
  - **All anonymous breakpoints** — Sets a breakpoint on an anonymous source.
  - **All named breakpoints —** Use this option when you do not know which form a particular page is using. You can set this breakpoint and go to the page. You should then disable the All named breakpoints option and refine your breakpoints to something more specific within that particular form, or the Debugger will stop at every breakpoint.

NOTE	If you enable both options, the Debugger checks all breakpoints.

- **View Cycle tab**: Select to set breakpoints associated with commonly used views (such as Checkin View, Checkout View, Get View, or Unlock View).

  You can set code breakpoints based on the view processing that occurs during process execution. The most commonly invoked view operations are listed in this dialog, and available on each view.

- **Form Cycle tab**: Select to set breakpoints associated with stages of form processing:

  You can set code breakpoints based on a designated stage of form processing. For information about the stages of form processing, see *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

The tutorial examples provided with Identity Manager illustrate how to use breakpoints. See "Using the Identity Manager IDE Tutorial: Debugging Forms, Rules, and Workflows" on page 78.

Using Watches

The Identity Manager IDE Debugger enables you to use *watches*, which can be any valid XPRESS expression. Watches serve two purposes:

- You can use simple `<refs>` to evaluate a specific set of variables.
- You can use more-complex XPRESS expressions to help you determine what will happen if you make certain changes to the project.

The Debugger evaluates watches in the current context when you first enter the Debugger, and everytime it suspends while stepping or at a breakpoint. The Debugger also re-evaluates watches whenever you add or modify a watch.

The tutorial provided with Identity Manager (`debugger-tutorial-workflow1.xml`) illustrates how to use watches. See "Example 1: Debugging a Workflow and a Rule" on page 79.

Stepping Through an Executing Process

Stepping through describes the sequential, deliberate analysis of an executing process' functions.

Terminology

Step into, *step over*, and *step out* are terms borrowed from debuggers of procedural programming languages, in which the execution order is implied by the structure of the language. However, in Identity Manager forms and workflows, the order in which elements occur in the code do not reflect the order in which they are executed.

Consequently, these terms have slightly different meanings when used in the Identity Manager IDE (or in the Business Process Editor (BPE)):

- **Step-Into**: Describes moving to the next point of execution on the current thread. *Step-Into* is always the smallest amount by which you can proceed through a process in the Source Editor.
- **Step-Over**: Describes moving from the current `begin` tag to the current `end` tag without stopping at an interim element. Stepping over permits you skip almost everything between a `start` and `end` tag. However, if the next point of execution does not occur between the `start` and `end` tags of the current element, debugging halts there instead.

  For example, in a workflow containing multiple active virtual threads, you can step to the `start` tag of an action, but the next element to be executed is a different action. In this case, the process stops executing at a different point. Thus, you avoid accidently skipping over a potentially significant element.
- **Step out**: Describes moving incrementally until the execution stack is one less than the current. Similar to step-over. If the next point of execution has a different parent execution stack, it will stop there instead.

TIP The following tips are provided to help you successfully step through an executing process:

- Set up stepping in the Debugger to be as granular as feasible in the context of your debugging task. This practice helps you avoid missing anything potentially critical to debugging.
- Stepping does not change the execution order of your program. The program's execution order is the same as it would be if the Debugger were not attached. You can skip seeing portions of the execution (but they still execute regardless).
- Use step-into when you want the smallest step possible through the code.
- Use step-over when you feel that no probable problem exists with the content between the `start` and `end` tag. The Debugger then skips this element although the code in between these tags still executes.

Table 1-8 provides a snapshot of how the Identity Manager IDE Debugger would proceed through this code sample:

```
<A>
   <B/>
</A>
<D/>
(A, B, and D are some xml elements)
```

Table 1-8 Example Debugging Process

Execution Order	Result
<A>, <B/>, </A>, <D/>	If you are clicking **step-into**, the Debugger highlights the lines in that execution order.
	If you are clicking **step-over**, the Debugger highlights <A>, </A> (skipping B), <D/>
<A>, <D/>, <B/>, </A>	If you are clicking **step-over**, you will see code lines in the order <A>, <D/>, <B/>, </A>. (Step-over is equivalent to step-into for this case.)

Debugging Forms

This section explains how to work with anonymous sources and provides instructions for configuring and launching the form tester utility.

For detailed information about how the Identity Manager Forms engine processes a form, see Sun Java™ System *Identity Manager Workflows, Forms, and Views*.

NOTE You can check the execution stack in the Call Stack window to see which pass is currently being processed.

Working with Anonymous Sources

When stepping through forms, the Debugger can identify *anonymous sources*. Anonymous sources are forms (or portions of a form) that are generated on the fly (such as Login forms and MissingFields forms), and do not correspond to a persistent form that resides in the Identity Manager repository.

Because these forms do not reside in the repository, and thus lack unique identifiers, you cannot set *individual* breakpoints on anonymous sources. However, you can step through an anonymous source if you set an All anonymous sources breakpoint (as described in "Setting Breakpoints" on page 67).

This setting enables the Debugger to stop execution at points that do not have direct XPRESS source. As a result, the Debugger breaks any time it encounters a line from an anonymous source.

Configuring and Launching the Form Tester

Identity Manager IDE provides a form tester that lets you test forms in an external browser. The form tester is integrated with the Debugger, so you can also troubleshoot your forms.

To test a form:

1. Use one of the following methods to launch the form tester:
   - Select the form node in the Projects window and select IdM > Test Form from the NetBeans menu bar.
   - Right-click on the form in the Projects window and select Test Form from the pop-up menu.
   - Right-click within a `<Form>` element or any of its children in the Source Editor and select Test Form from the pop-up menu.

NOTE A warning message displays to let you know that when Identity Manager IDE uploads your form to the repository it will replace (overwrite) the existing copy. You must indicate whether to continue or stop the test process.

2. When prompted, log into Identity Manager.

TIP To avoid having to log in every time you test a form, use the following steps to modify the System Configuration object's `allowInterAppAuthenication` property:

1. Select IdM > Repository > Explore.
2. From Explore Repository window, expand the Generic Objects node and double-click System Configuration to download the object.
3. Double-click `System Configuration.xml` to view it in the Source Editor, and then scroll down (or use Edit > Find) to locate the `allowInterAppAuthentication` attribute.
4. Change the attribute's value to **`true`** and save your change.

Identity Manager IDE launches a new Identity Manager session in a browser window, and displays your form in the context of the Identity Manager Administrator user interface. In addition, information about the uploaded form displays in the Identity Manager IDE Repository Output window.

NOTE Identity Manager provides a tutorial (`debugger-tutorial-workflow1.xml`) to help you understand how to debug forms. See "Using the Identity Manager IDE Tutorial: Debugging Forms, Rules, and Workflows" on page 78 for instructions.

Testing Rules

Identity Manager IDE provides a rule tester that allows you to verify standalone rules and library rules as you edit them in the Source Editor. You provide values for any rule argument in the Source Editor and then "run" the rule. (You can also embed trace statements.)

You can use one of the following methods to launch the rule tester:

- Select the rule node in the Projects window and select IdM > Test Rule from the NetBeans menu bar.
- Right-click on the rule in the Projects window and select Test Rule from the pop-up menu.
- Right-click within a `<Rule>` element or any of its children in the Source Editor and select Test Rule from the pop-up menu.

The following examples are provided to illustrate how the rule tester works.

- Example 1: Testing Standalone Rules
- Example 2: Testing Library Rules

Example 1: Testing Standalone Rules

This example illustrates how to test a simple, standalone rule:

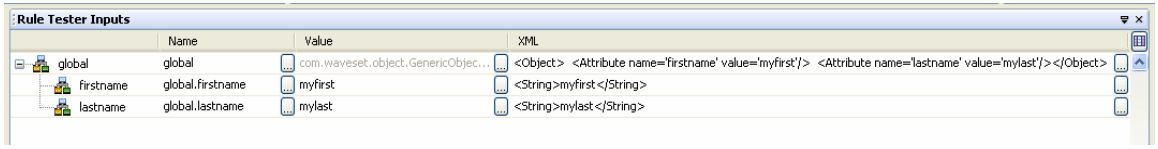
1. Right-click Custom Identity Manager Object > Repository > Explore to view objects in the repository.
2. In the Explore Repository dialog, expand the Rules node and double-click `Accountname First dot Last` to open the rule.
3. Right-click in the Rule Tester Inputs window and select Add value from the menu.
4. When the New Value dialog displays, enter the following information, and then click OK:
    - **Name**: `global.firstname`
    - **Type**: `String`
    - **Value**: `myfirst`

5. Right-click in the Rule Tester Inputs window again, and enter another new value with the following information:

   - **Name**: `global.lastname`
   - **Type**: `String`
   - **Value**: `mylast`

   Your Rule Tester Inputs window should look similar to Figure 1-34.

Figure 1-34 Rule Tester Inputs Window



6. You can now test the rule.

   Select IdM > Test Rule from the NetBeans menu bar or right-click in the Source Editor window and select Test Rule from the pop-up menu.

   NOTE A message displays to warn you that the rule test will automatically save the rule and publish it to the repository. Click Yes or Always to proceed.

   You should see output displayed in Rule Tester Output window that is similar to the following:

   ```
   <String>myfirst.mylast</String>

   <String>myfirst.mylast</String>

   <String>myfirst.mylast</String>
   ```

7. Next, try testing the rule and running the Debugger at the same time. If the Debugger is not already running, click the Debug Main Project button on the main menu bar.

8. Go the Source Editor window, click in the margin next to the `<Rule>` tag in the XML, and add a breakpoint.

9. Select Window > Debugging > Breakpoints to open the Breakpoints window.

10. In the Source Editor, right-click anywhere in the `<Rule>` element and select Test Rule from the pop-up menu.

    The Debugger will stop at your breakpoint. (See the result in the Breakpoint window.)

11. Click the Step Into button  seven times to step through your rule, observing the results in the Local Variables window.

12. Click Continue and observe the results in the Rule Tester Output window.

Example 2: Testing Library Rules

Testing library rules is very similar to testing standalone rules:

1. If necessary, download the Alpha Numeric Rules library from the repository and double-click the node to open this library rule in the Source Editor.

2. Expand the Alpha Numeric Rules nodes until you can double-click the `stringToChars` rule.

   The `testStr` argument displays in the Rule Tester Inputs window because this rule declares a formal argument.

3. Give the `testStr` argument a value (right-click on the argument name in the Rule Tester Inputs window, select Add value, and enter a value in the New Value dialog).

4. Right-click in the Source Editor and select Test Rule to run the rule and observe the results in the Rule Tester Output window.

NOTE Identity Manager provides a tutorial (`debugger-tutorial-workflow1.xml`) to help you understand how to debug rules. See "Using the Identity Manager IDE Tutorial: Debugging Forms, Rules, and Workflows" on page 78 for instructions.

Debugging Workflows

Workflows are executed by a single Java thread, and are represented in the Call Stack window by a single Java thread. However, within a workflow, each activity becomes its own virtual thread.

During workflow execution, the workflow engine cycles through a queue of virtual threads. Each virtual thread is in one of the states described in the following table.

Table 1-9 Virtual Thread States

Workflow Activity State	Definition
ready	Identifies an activity that has just been transitioned to. (This state is very temporary, as actions typically start executing immediately after being designated ready.)
executing	Identifies an activity that contains one or more actions that are currently being executed or have yet to run. This is a logical state, which does not mean that the Java thread is currently executing it. The action currently being executed is always in bold/normal font. Actions that are not being executed are displayed in italics.
pending outbound	Identifies an activity after all actions within an activity have been run, it goes to the pending outbound state. In this state, it awaits an outbound transition to be taken. In the case of an or-split, it is in this state until one transition is taken. In the case of an and-split, it will be in this state until all transitions whose conditions evaluate to true are taken.
inactive	Identifies an activity in which all transitions have been taken.
pending inbound	Identifies a virtual thread whose activity is an and-join. That is, one transition to this virtual thread has occurred, but the process is still waiting for other transitions.

After all transitions have completed, the workflow process subsequently begins executing.

To validate your workflow revisions after making changes, select the object or process in the Projects window, and then select Tools > Validate to test it. Check the Output window for messages.

NOTE Identity Manager provides a tutorial (`debugger-tutorial-workflow1.xml`) to help you understand how to debug workflows. See "Using the Identity Manager IDE Tutorial: Debugging Forms, Rules, and Workflows" on page 78 for instructions.

Using the Identity Manager IDE Tutorial: Debugging Forms, Rules, and Workflows

Identity Manager provides a tutorial (`debugger-tutorial-workflow1.xml`) to help you learn how to use the Debugger with forms, rules, and workflows. This tutorial contains sample forms, rules, and workflows that are used throughout this section.

CAUTION *Do not* enable this tutorial in your production environment.

This section is organized as follows:

Getting Started

To use the `debugger-tutorial-workflow1.xml` tutorial,

1. Use one of the following methods to import the `debugger-tutorial-workflow1.xml` file:
   - From Identity Manager, select Configure > Import Exchange File. Type **`sample/debugger-tutorial.xml`** into the File to Upload field.
   - From the Console, enter:

     `import -v sample/debugger-tutorial.xml`

     If the file import successfully, you will see confirmation that the following files were loaded:

     ```
     debugger-tutorial-workflow1
     debugger-tutorial-workflow2
     compute-full-name
     ```
2. Download `debugger-tutorial-workflow1` and `debugger-tutorial-workflow2` from the Repository to your project. (See "Getting Objects from the Repository" on page 43 for instructions.)
3. After restarting the application server, select Run > Debug Main Project from the main menu bar to launch the Identity Manager IDE Debugger.

Example 1: Debugging a Workflow and a Rule

This example illustrates how to debug a simple workflow and rule that uses the workflow, including how to step-into and step-through workflow debugging and rule execution.

To complete this exercise you must perform the following steps:

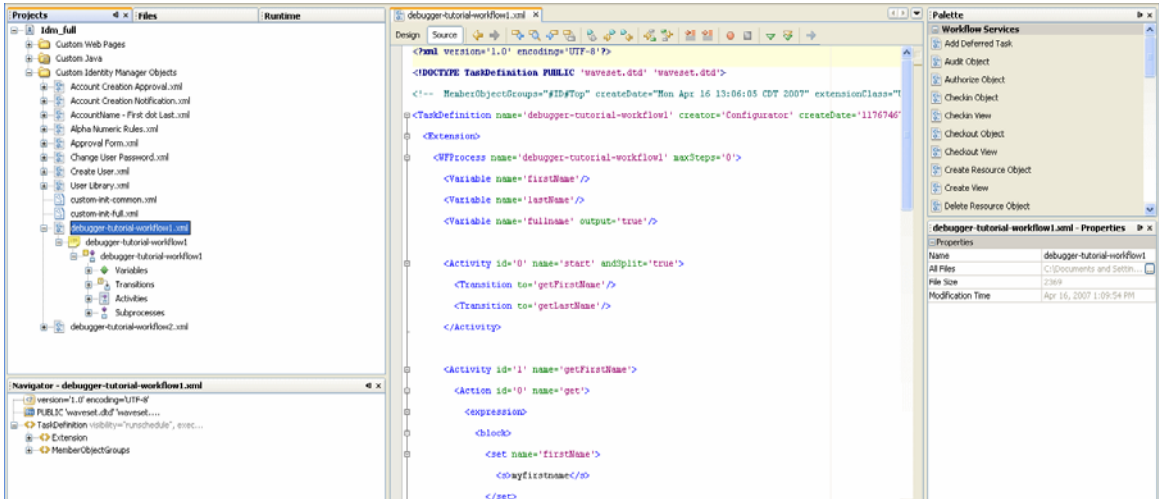
1. Launch the process.
2. Start execution.
3. Step through the `getFirstName` thread.
4. Step into and over the `getlastname` thread.
5. Step into `computefullname` processing.
6. Step through rule processing.
7. Conclude workflow processing.

Step 1: Launch the Process

To launch the workflow debugging process:

1. In the Project window, expand the `debugger-tutorial-workflow1.xml` node.
2. Double-click the `debugger-tutorial-workflow1.xml` node to view the XML in the Source Editor. (If necessary, click the Source button.)

Figure 1-35 Open debugger-tutorial-workflow1.xml Source



3. Set a breakpoint at the start of the workflow by clicking once in the margin, to the left of the `<WFProcess>` tag.
4. If necessary, click the Debug Main Project button on the main menu bar to launch the Identity Manager IDE Debugger.
5. Log in to Identity Manager, and then select Server Tasks > Run Tasks.
6. When the Available Tasks page displays, click debugger-tutorial-workflow1.xml in the Name column.

   Check the Identity Manager IDE Source Editor, and you should see that debugging halted at your breakpoint.

   Also, note the following:

   - At the top of the Call Stack window, you should see `Thread [`*thread name*`] (suspended)`, which indicates that this workflow is currently being run by the thread of the given name, and that it is suspended at the breakpoint you set.

     Below the Thread is your execution stack. This stack is an upside-down stack trace, with the calling function on top and the called function at the bottom. (It is upside down compared with how most debuggers represent execution stacks.)

     The top most frame in the stack says Checkin View (ProcessViewer), which indicates that the workflow is being called by the `checkinView` method of the ProcessViewer. Because you do not have access to the Java source code for this stack frame, clicking on it will not display new information. However, the stack frame does provide context about where the workflow is being launched from.

     The next frame in the stack is highlighted because it corresponds to the current point of execution, which is the beginning of the workflow process (`<`*`WFProcess`*`>`).

   - In the Local Variables window, you should see a list of all variables that are currently in scope at the current point of execution, including

Table 1-10 Variables in Scope

Variable	Description
Last Value	Result of the last evaluation
Interactive	Variable passed in by the view as an input to the process.
WF_CASE_OWNER	Implicit workflow variable

Table 1-10 Variables in Scope *(Continued)*

Variable	Description
fullname	Variable declared in the workflow using `<Variable>` declarations
WF_CONTEXT	Implicit workflow variable
WF_CASE_RESULT	Implicit workflow variable
firstName	Variable declared in the workflow using `<Variable>` declarations
lastName	Variable declared in the workflow using `<Variable>` declarations

Now you are ready to start execution. Go to the next section for instructions.

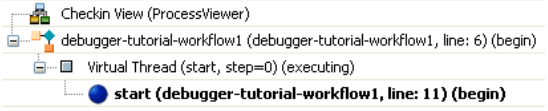
Step 2: Start Execution

To start execution:

1. Click the Step-Into button  on the Identity Manager IDE main menu bar.

   The Debugger moves to the start activity. Notice that the execution stack contains a `Virtual Thread [start, step=0] (executing)`, which indicates there is a virtual thread for the start activity that is currently in the executing state.

Figure 1-36 Virtual Thread for the Start Activity



2. Double-click on the `debugger-tutorial-workflow1` frame (two levels up) to highlight the `WFProcess`, which shows you the caller location.
3. Double-click on the bolded entry in the Call Stack window to return to the current line.
4. Click Step-Into again.

   The Debugger moves to the `</Activity>` line in the Source Editor, and if you look in the Call Stack window, the `Virtual Thread [start, step=0]` is now pending outbound.

You can now proceed to Step 3, in the next section.

Step 3: Step Through the getFirstName Thread

To step-through the `getFirstName` thread:

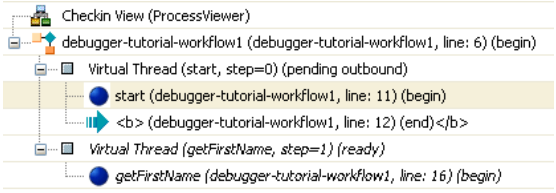
1. Click Step-Into.

   The Debugger highlights the transition to `getFirstName`.

2. Click Step-Into again.

   Check the Call Stack window again, and notice that the Debugger created a new Virtual Thread for `getFirstName` as a result of this transition, and this Virtual Thread is currently in the ready state.

Figure 1-37 Virtual Thread for the Start Activity



   Note that `Virtual Thread [start, step=0]` is still pending outbound because it is an `and-split` operation, and must take all possible transitions.

3. Click Step-Into again.

   The Debugger jumps to the `getFirstName` activity in the Source Editor, and in the Call Stack window, the state changes from ready to executing.

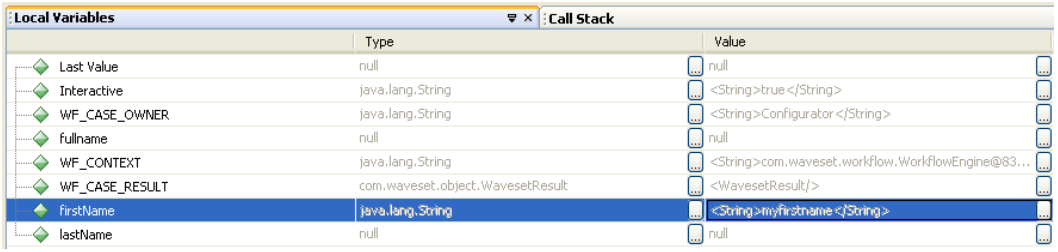
4. Click Step-Into again.

   The Debugger moves to the `get` action.

5. Now, click Step-Into three more times or until the Debugger reaches the `</set>` tag in the Source Editor.

   Check the Local Variables window and note that the `firstName` value is set to `<String>myfirstname</String>` as a result of the `</set>`.

Figure 1-38 New firstName Value



6. Add a watch expression:

   a. Open the Watches window (Window > Debugging> Watches).

   b. Right-click in the window and select New Watch from the menu.

   c. When the New Watch dialog displays, type the following XPRESS statement into the Watch Expression field, and then click OK.

      `<ref>firstName</ref>`

   The Debugger evaluates the watch expression, and its value should be `<String>myfirstname</String>`, which is the value you just set in Step 5.

Step 4: Step Into and Over the getLastName Thread

Use the following procedure to step-into and over the `getLastName` thread:

1. Click Step-Into three more times or until the Debugger reaches the `</Activity>` line for `getFirstName` in the Source Editor.

   Note that `Virtual Thread (getFirstName, step=1)` is now pending outbound in the Call Stack window.

2. Click Step-Into.

   The Debugger returns to the `Virtual Thread (start, step=0)`, and is about to process the transition to `getLastName`.

3. Click Step-Into.

   `Virtual Thread (start, step=0)` becomes inactive because all transitions have been processed. As a result of this transition, `getLastName` is now in the ready state.

4. Click Step-Into.

   At this point, `Virtual Thread (start, step=0)` goes away because it is inactive, and the Debugger moves to the `Virtual Thread (getLastName, step=2)`, which is now in the executing state.

5. Click the Step-Over button  to skip to the end of `getLastName`.

   Check the Local Variables window, and the `lastName` variable should be set to `<String>mylirstname</String>`. Both the `getFirstName` and `getLastName` virtual threads are pending outbound.

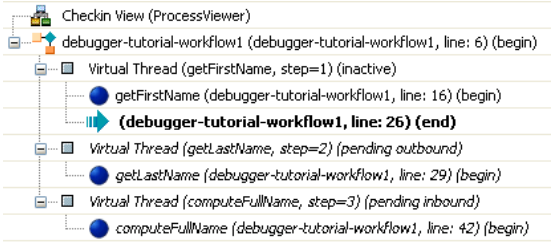
6. Click Step-Into.

   Check the Source Editor and note that the Debugger is on the Transition from `getFirstName` to `computeFullName`.

7. Click Step-Into.

   In the Call Stack window, `getFirstName` becomes inactive and a new `Virtual Thread, (computeFullName, step=3)` is created.

Figure 1-39 New computeFullName Virtual Thread



   Note that this thread is in the pending inbound state because it is still waiting on the inbound transition from `getLastName`. (The wait occurs because it is an `and-join` operation. If it were an `or-join` operation, process status would immediately go to ready.)

8. Click Step-Into.

   The Debugger is now on the transition from `getLastName` to `computeFullName`.

Step 5: Step Into computeFullName Processing

Use the following procedure to step-into `computeFullName` processing:

1. Click Step-Into.

   The `Virtual Thread (computeFullName, step=3)` goes from pending inbound to ready because of this transition.

2. Click Step-Into.

   `Virtual Thread (computeFullName, step=3)` is now executing.

3. Click Step-Into five more times.

   If you check the Source Editor, you should see that the Debugger is now on the `</argument>` tag for `firstName`, and that Last Value in the Local Variables window is now `<String>myfirstname</String>`. This value is passed for the `firstName` argument.

Step 6: Step Through Rule Processing

To Step Through rule processing:

1. Click Step-Into three more times.

   The Debugger steps into the `compute-full-name` rule.

2. In the Call Stack window, click the frame to move up one frame.

   The `<rule>` call in `debugger-tutorial-workflow1` is highlighted to indicate from where the rule is being called.

3. Double-click the bolded line to re-select that line.

4. Click Step-Into three more times or until the Debugger reaches the `</ref>` tag.

   Now check the Local Variables window, and the Last Value entry is `<String>myfirstname</String>`, which is the result of the `<ref>firstName</ref>`.

5. Click Step-Into three more times or until the Debugger reaches the `</concat>` tag.

   The Last Value entry is now `<String>myfirstname mylastname</String>`, which is the result of the `<concat>` expression.

6. Click Step-Into twice more and the Debugger returns to the `</rule>` tag.

Step 7: Conclude Workflow Processing

To conclude workflow processing:

7. Click Step-Into until you reach the `</set>` element.

   The `fullname` variable is updated to `<String>myfirstname mylastname</String>`.

8. Click Step-Into twice more.

   `Virtual Thread (computeFullName, step=3)` is now pending outbound.

9. Click Step-Into four more times.

   Notice that `end` goes to ready, then executing, and then the Debugger reaches the `</WFProcess>` tag, which indicates that the process is now complete.

10. Click Step-Into.

    In the Call Stack window, `After Checkin` View displays, indicating that the checkin-view operation that launched this workflow is now complete.

11. Click the Continue button on the Identity Manager IDE main menu bar to resume execution.

    If the browser request has not timed out, the Task Results diagram with the process diagram displays.

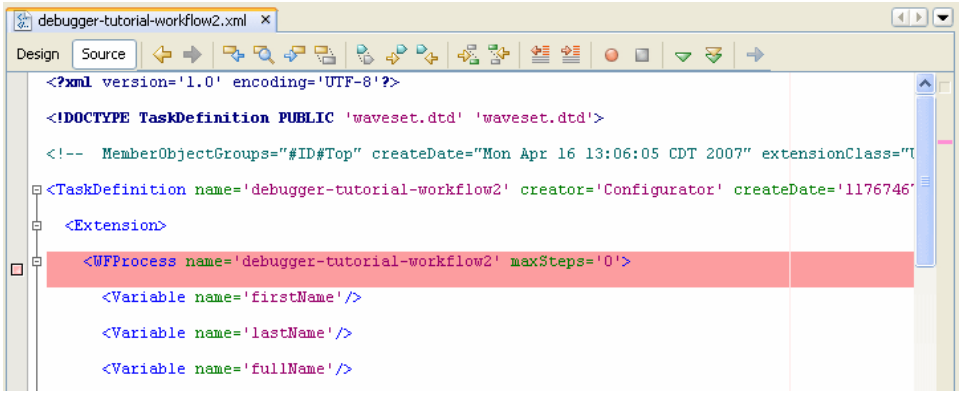
Example 2: Debugging a Workflow Containing a Manual Action and a Form

This example illustrates how to debug a sample workflow containing a manual action and a form. Use `debugger-tutorial-workflow2` tutorial file and perform the following steps:

1. Double-click `debugger-tutorial-workflow2.xml` to view the XML in the Source Editor.
2. Set a breakpoint on the `<WFProcess...>` tag.
3. If necessary, click the Debug Main Project button  on the Identity Manager IDE main menu bar.
4. Log in to Identity Manager, and navigate to Server Tasks > Run Tasks.
5. When the Available Tasks page displays, select `debugger-tutorial-workflow2` from the Name column.

Notice that the Debugger has stopped at the breakpoint you set.

Figure 1-40 New Breakpoint



6. Click Step-Into six times or until the Debugger marks `<ManualAction... name='getNameAction'>`.
7. Click Step-Into.
8. When the Stepping into Manual Action dialog displays to explain that form processing occurs in a different thread, click Yes or Always.
9. Set a breakpoint on the `<Form>` tag to see the processing occur.

   When the form processing is complete, the workflow continues execution in a separate thread. Consequently, you must set a breakpoint on the `</ManualAction>` to observe workflow processing after the form has completed processing.

   Note that the Debugger set breakpoints on the `<Form>` and `</ManualAction>` tags as indicated. In addition, the Call Stack window indicates After Checkin View, and you have stepped out of the workflow process because workflow processing proceeded as far as possible (until the manual action completes).
10. Click Continue, and the Debugger stops processing at the breakpoint set on the `<Form>` element.

Note the following entries in the Call Stack window:

- **Derivation** — Indicates that form execution is on the derivation pass.
- **Checkout View (WorkItem:...)** — Indicates that processing is occurring in the context of a checkout view for the given work item.
- **ManualAction forms** — Operate against the work item view and manipulate workflow variables through the variables object. Expand the variables object to see the non-null workflow variables.

11. Because this form contains no `<Derivation>` expressions, click Continue to proceed to the next processing phase. The HTML Generation (root component) pass of form processing begins.

HTML Generation Phase (Root Component)

To generate HTML for the root component:

1. Click Step-Into twice.

   The Debugger just processed the title `<Property>` element, and the Last Value entry in the Local Variables window contains the value of this property.

2. Click Step-Into three more times.

   The Debugger skips the form fields and goes directly to the `</Form>` element because this pass focuses only building the root component of the page.

3. Click Continue to begin the HTML Generation (subcomponents) pass of form processing.

HTML Generation (Subcomponents)

To generate HTML for the subcomponents:

1. Click Step-Into 13 times or until the Debugger reaches the `</Form>` tag.

   The Debugger iterates over each of these fields and evaluates their display properties.

2. Click Continue.

   The Debugger displays No suspended threads because execution has resumed. Control has now returned to your Identity Manager browser window.

3. Return to the Identity Manager browser window, and enter your first and last name as prompted, and click Save.

   Return to your Debugger frame, and note that the Debugger is now suspended on your breakpoint.

4. Expand the Variables subtree in the Local Variable window, and note that the names you just entered are displayed as the `firstName` and `lastName` values.

   The Debugger is currently in the Confirmation phase of form processing.

Confirmation

Because this form has no confirmation fields, no processing occurs.
Click Continue to begin the Validation phase of form processing.

Validation and Expansion

Because this form contains no validation expressions, no obvious processing occurs.

1. Click Continue to skip the Validation phase and proceed to the Expansion phase of form processing.

2. Click Step-Into six times.

   The Debugger moves to the `<rule>` tag of the `<Expansion>` of the `variables.fullName` field.

3. Click Step-Into five times, and the Debugger steps into the `<Rule>` element.

4. Click Step-Into seven times or until the Debugger reaches the `</Rule>` element.

   The Last Value contains the full name.

5. Click Step-Into again and processing resumes in the form.

6. Click Step-Into again.

   The top-level `variables.fullName` has the value of the Expansion expression you just ran. This value is a top-level entity rather that a child of the `variables` data structure because during form processing, form outputs are kept in their own temporary `form_outputs` data structure, with path expressions flattened.

   After form processing, form outputs are assimilated back into the view. In the implicit variables `form_inputs` and `form_outputs`, `form_inputs` shows the unmodified workitem view, and `form_outputs` shows the output fields that are assimilated back into the view after form processing completes.

   In general, `form_inputs` identifies the view, and `form_outputs` contains data to be assimilated back into the view. However, not all forms are necessarily tied to a view (for example, active sync forms). The form engine is a general data mapping engine, mapping from form inputs to form outputs. The view handler is responsible for passing the view into the form engine and assimilating the outputs back into the view.

7. Click Continue.

   The Debugger reaches the `</ManualAction>` breakpoint that you set previously when the Debugger stepped into the Manual Action. The `firstName` and `lastName` variables are the values that you entered. The `fullName` value is the result of the Expansion expression that just ran.

8. Click Step-Into five times until you get to `<ManualAction... name='displayNameAction'>`.

9. Click Step-Into again. (If prompted, click Yes or Always.)

10. Click Continue.

    The Debugger is now on the Derivation pass for `displayNameForm`.

Derivation and HTML Generation (Root Component)

To complete the derivation and HTML generate phase

1. Click Continue to begin the HTML Generation (root component) processing for `displayNameForm`.

2. Click Step-Into eight times or until the Debugger reaches the `</Property>` element for `subTitle`.

3. Click Continue twice.

The Debugger displays the following message:

```
No suspended threads because execution has resumed. Control has now
returned to the browser window.
```

4. Return to your Identity Manager browser window.

   The information displayed is the same you entered.

5. Click Save and return to your Debugger frame.

   The Debugger is now on the Confirmation pass, processing the `displayNameForm`.

Validation and Expansion

To begin validation and expansion,

1. Click Continue to begin the Validation pass.
2. Click Continue to begin the Expansion pass.
3. Click Continue again.

   The Debugger is now on the `</ManualAction>` tag because the manual action is complete. At this point, workflow processing has resumed.

4. Click Step-Into five times or until the Debugger reaches the `</WFProcess>` tag, indicating that the workflow has completed execution.
5. Click Continue.
6. Return to the Identity Manager window and you should see the workflow process diagram. Click OK.

Example 3: Debugging the Tabbed User Form and Refresh View

This sample debugging procedure illustrates how Debugger breakpoints apply regardless of where you launch a form or workflow.

To complete this procedure, you must perform the following steps:

1. Set a breakpoint.
2. Create a new user.
3. View before refresh view results.
4. View after refresh view results.

5. Step through the form.
6. Finish processing the form.

Setting a Breakpoint

To set a breakpoint:

1. Right-click in the Breakpoints window and select New Breakpoint from the menu.
2. When the New Breakpoints dialog displays, select XPRESS from the Debugger menu and then select the View tab.
3. Enable the Refresh View check box.

   The Debugger will now execute a breakpoint whenever a view is refreshed during execution.

Creating New User

To create a new user:

1. In Identity Manager, select the Accounts tab, and then select New User from the top, left drop-down menu.
2. When the Create User page displays, enter a first name and a last name (for example, **`jean faux`**).
3. Click a different tab to trigger a refresh view operation.

   Note that Identity Manager is now suspended because it hits a breakpoint.

Viewing Before Refresh View Results

Return to the Identity Manager IDE and note the following:

- The Source Editor is now suspended on the Refresh View breakpoint that you set.
- The Call Stack window lists Before Refresh View, which indicates the state of the view just before the refresh operation occurred.
- The Local Variables window displays the view just before it was refreshed.

In the Local Variables window, expand the global subtree and locate the `firstname` and `lastname` values that you typed in the form. Note that the `fullname` is currently `null`.

Viewing After Refresh View Results

To view the After Refresh view results,

1. Click Continue.

   The Call Stack window lists After Refresh View, indicating it now displays the state of the view just after the refresh occurred. Note that the `fullname` value is now `jean faux`.

2. Click Continue again.

   The form resumes execution. Return to the Identity Manager browser window and change First Name to **`jean2`**. Click a different tab to trigger another refresh.

   Back in the Identity Manager IDE Source Editor, the form processing is suspended at Before Refresh View.

Stepping Through the Form

To Step Through the form,

1. Click Step-Into to reveal the fullname expansion in execution.

   The Call Stack window lists Before Expansion, which indicates that the form variables have not been expanded.

2. Click Step-Into again.

   The Call Stack window now lists Before Expansion, `iteration=0`, indicating that you will see the form variables before the first Expansion pass.

3. Click Step-Into again.

   The Call Stack window lists an anonymous source (Tabbed User Form (Anonymous, line: 3)(begin)). The anonymous source is a wrapper form created on the fly and it is related to the MissingFields form.

4. Click Step-Into two more times until you reach the beginning of Tabbed User Form.

5. Continue to click Step-Into until you reach `<Field name='global.fullName'>` (Approximately 20 to 30 step-into operations.)

6. Click Step-Into 15 times or until you have reached the `</Field>` element.

   While stepping, notice that the Last Value entry at the `</concat>` tag is `jean2 faux`, and that the `form_outputs` value is `global.fullname: jean2 faux`.

Complete Form Processing

To complete form processing:

1. Click Step-Out seven times.

   At this point, the Call Stack window should indicate:

   ```
   Refresh View (User)
   After Expansion
   ```

   The Local Variables window displays the state of the form variables after all expansions have run.

2. Click Step-Out again.

   You have now reached After refresh view. The Local Variables window now displays the view variables.

3. Expand the global subtree.

   Note that `fullname` is now `jean2 faux`.

4. Click Continue.

Debugging Java and XPRESS

The following example illustrates how to create and compile custom Java code and how to use both the XPRESS debugger and Java debugger at the same time.

NOTE The Identity Manager Project (Remote) does not support Java debugging.

TIP Before starting this example, turn off the NetBeans HTTP monitor to remove the extra window:

1. Expand the Servers node in the Runtime window, right-click Bundled Tomcat, and select Properties from the pop-up menu.
2. When the Server Manager dialog displays, disable the Enable HTTP Monitor box, and then click Close.

1. Perform the following steps to create custom Java code:

   a. In the Projects tab, right-click Source Packages and select New > Java Class from the pop-up menu.

   b. Enter **`TestClass`** in the Class Name field and **`testpackage`** in the Package field.

   c. When the `testpackage` node displays, add the following method:

```
public static String concat(String s1, String s2)
{
   return s1+s2;
}
```

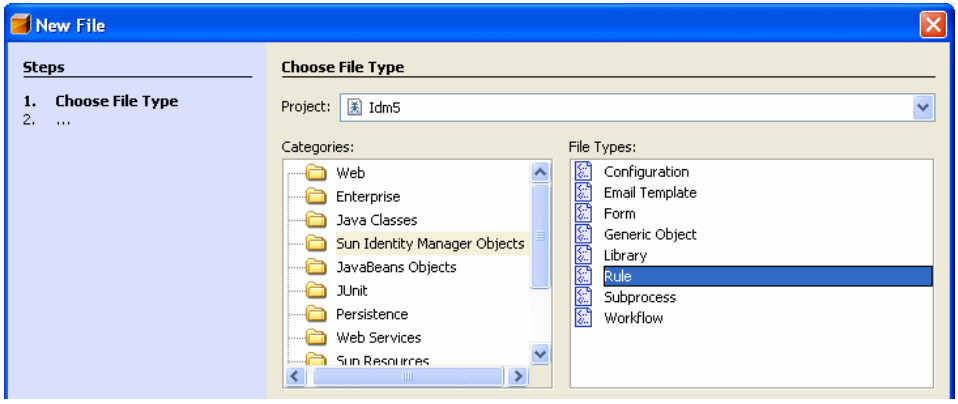
   d. Save the class.

2. Perform the following steps to create a rule that calls the Java code:

   a. In the Project window, right-click Custom Identity Manager Objects and select New > File/Folder from the pop-up menu.

   b. When the New File dialog displays, select Sun Identity Manager Objects from the Categories list, and then select Rule from the File Types list.

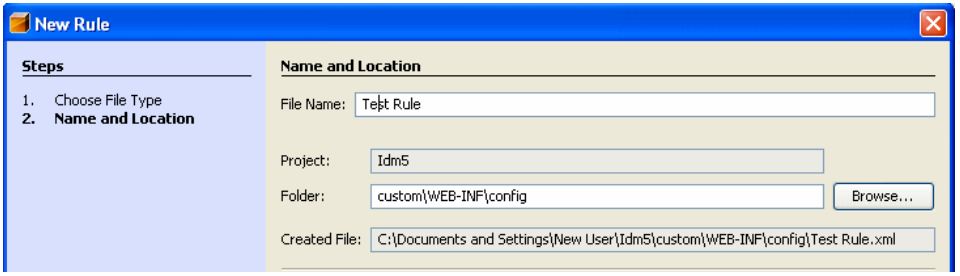
Figure 1-41 Choosing the Rule File Type



   c. When you are done, click Next.

d. When the New Rule screen displays, enter **`Test Rule`** into the File Name field, and then click Finish.

Figure 1-42 Choosing the Rule File Type



The Projects window updates to display `Test Rule.xml`.

e. Select `Test Rule. xml` and then click the Source tab in the Editor window to view the XML.

f. If necessary, change `<Rule name='New Rule'` to `<Rule name='`**`Test Rule`**`'`.

g. Insert the following call to your method and then save the rule:

```
<invoke class='testpackage.TestClass' name='concat'>
        <ref>arg1</ref>
        <ref>arg2</ref>
</invoke>
```

3. Perform the following steps to build the custom Java code and start the application server:

a. Right-click on the project node and select Debug Project.

Wait while Identity Manager IDE builds the Web application and starts the server (which takes a few minutes).

b. When prompted, enter your password.

If you check the log file, you will see that you rule was automatically uploaded.

c. When a browser window displays, you can close it.

   If you want to disable the browser, right-click the project and select Properties. In the Project Properties window, Run and uncheck Display Browser on Run.

4. Set a breakpoint in the Java code and one in the XPRESS code:

   a. In `Test Rule.xml`, set a breakpoint by clicking in the margin to the left of the `<Rule` tag.

   b. In `TestClass.java`, set a breakpoint by clicking in the margin to the left of the `return s1+s2;` statement.

5. Use the rule tester to call both breakpoints:

   a. Select Window > Rule Tester Inputs from the NetBeans menu bar to open the Rule Tester Inputs window.

   b. Select `Test Rule.xml` in the Projects window.

   c. Right-click in the Rule Tester Inputs window and select Add value. When the New Value dialog displays, enter the following information, and then click OK:

      - Name: **`arg1`**
      - Type: **`String`**
      - Value: **`myvalue1`**

   d. Right-click in the Rule Tester Inputs window again and select Add value. When the New Value dialog displays, enter the following information, and then click OK:

      - Name: **`arg2`**
      - Type: **`String`**
      - Value: **`myvalue2`**

   e. Expand the `Test Rule.xml` node, right-click Test Rule, and then select Test Rule from the pop-up menu.

   f. When a message displays asking you to confirm whether to save the rule, click Always.

      At this point, you have reached the breakpoint on your rule.

g. Click the Step-Into button  six times to reach the breakpoint in your Java code.

Note that you are now in the Java Debugger, and that your Java variables are now displayed in the Local Variables window.

NOTE	When you are debugging Java and XPRESS at the same time, there are actually two debuggers running at the same time and they are not aware of each other. Both debuggers have their own breakpoints and their own stepping-into states. Step-into will not step from XPRESS code into Java code unless there is a breakpoint in the Java code.

h. Click Continue.

Note that you have returned to the `</invoke>` tag in the XPRESS Debugger. You should also see that Last Value shows `myvalue1myvalue2` as the invoke result.

NOTE	When leaving the Java code, always click Continue to resume executing the Java debugger. Debugging performance becomes very sluggish if you leave the Java debugger in a stepping state.

i. Click Continue again to see the rule results in the Rule Tester Output window.

6. Modify the rule as follows, and then rerun the rule:

   a. Change `<ref>arg1</ref>` to `<s>myprefix</s>`.

   b. Right-click the rule and select Test Rule.

   c. Click Continue twice and view the new results in the Rule Tester Output window.

7. Modify the Java code as follows, and then re-run:

   a. Change `return s1+s2;` to `return s1+s2+"mysuffix";`

   b. Because you changed the Java code, you must rebuild and redeploy. Right- click the project and select Debug Project.

c. Right-click the Test Rule node and select Test Rule.

d. Click continue twice and you should now see the following results in the Rule Tester Output window:

```
<String>myprefixmyvalue2mysuffix</String>
```

Stopping the Debugger

To stop the Identity Manager IDE Debugger, select Run > Finish Debugging Session from the main menu bar.

Disabling the Debugger

You should always disable the Debugger when you are finished debugging to prevent someone from accidently connecting it to an application server in your production environment.

To disable the Debugger, use the following steps:

1. From Projects tab, expand the Generic Objects node and double-click System Configuration to view the XML in the Source Editor window.
2. Scroll down (or use Edit > Find) to locate the `serverSettings.default.debugger.enabled` attribute and change the value to **`false`**, as shown in the following example:

```
<Attribute name='serverSettings'>
        <Object>
          <Attribute name='default'>
            <Object>
              <Attribute name='debugger'>
                <Object>
                  <Attribute name='enabled'>
                    <Boolean>false</Boolean>
                  </Attribute>
```

3. Select File > Save from the main menu bar to save your change.
4. Restart your application server.

Running the Debugger Outside a Test Environment

When you encounter problems in production that require debugging, it is best to try to reproduce and debug those problems in a test environment. Setting breakpoints in the Debugger can quickly bring down the application server in your production environment, where a large volume of traffic occurs. You might also block others from using the system (depending where you set the breakpoints).

If you cannot debug in a separate test environment, follow this procedure:

1. Divert all live traffic to a subset of your cluster by taking one of the nodes in your cluster offline. (For the purpose of this task, call this node *server-a*.)
2. Use the Identity Manager IDE to edit the System Configuration object by setting the `SystemConfiguration serverSettings.server-a.debugger.enabled` property to `true`.
3. Restart server-a so that the change to the System Configuration property setting can take effect.
4. Change your project settings; specifying the proper host (server-a), port, context path, user, and password.
5. Start the Debugger.
6. When you have finished debugging, set `serverSettings.server-a.debugger.enabled` to `false` and restart server-a to prevent the Debugger from connecting to your live production environment.
7. Reintegrate server-a into your on-line cluster.

Uninstalling Identity Manager IDE from NetBeans

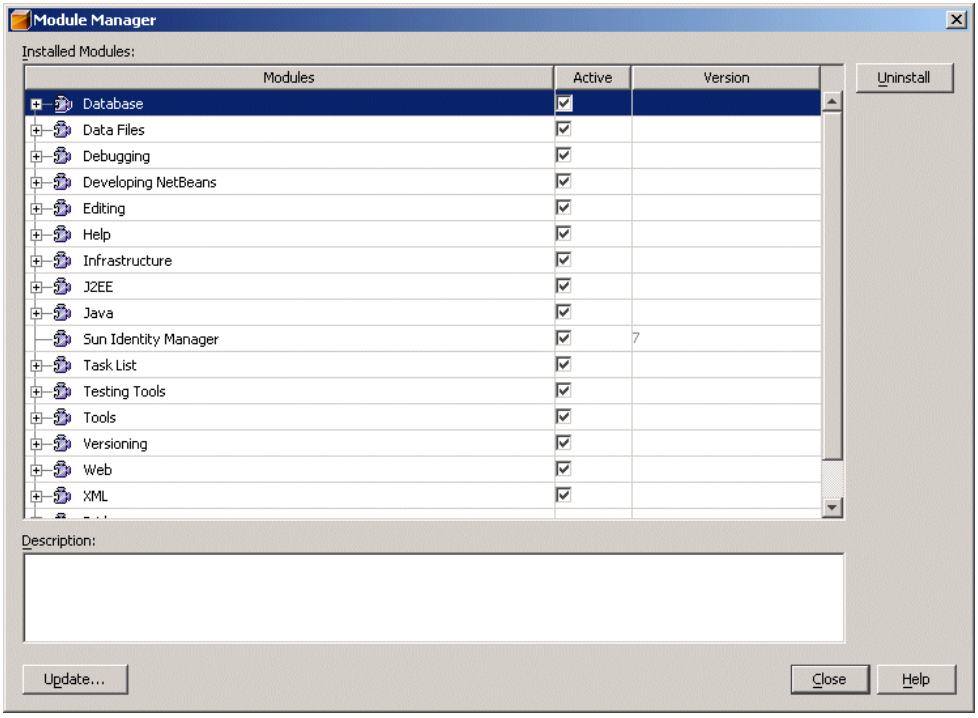
If for any reason, you want to uninstall the Identity Manager IDE module from NetBeans, perform the following steps:

1. If necessary, open NetBeans.
2. Select Tools > Module Manager from the NetBeans menu bar.

   The Module Manager dialog displays (Figure 1-43), listing all of your installed modules. The Active column indicates which of these modules are enabled.

NOTE The Identity Manager IDE allows you to disable the modules you do not need to minimize start-up time and save memory. Disabled modules are not deleted from your installation directory, but are simply ignored by the Identity Manager IDE. You can re-enable disabled modules at any time.

Figure 1-43 Selecting the Module to Uninstall



3. Select Sun Identity Manager from the list of modules.
4. Click Uninstall.
5. A pop-up displays, asking you to confirm that you want to uninstall the Sun Identity Manager module. Click OK to proceed with the uninstall process.
6. When the process is complete, close the Module Manager.

Troubleshooting Identity Manager IDE

This section provides information you might need to troubleshoot issues with Identity Manager IDE.

Unable to Delete Errors

Unable to Delete error messages can occur in the following circumstances:

- If you select IdM > Manage Embedded Repository while the server is running, the following error message will display:

  ```
  Unexpected Exception: 'Unable to delete
  [NetbeansHome]\[Project]\idm-repository\idm.data'. See the logs for
  details.'
  ```

  To correct this error, shut down the server and try again.

- If you try to performing a “clean” while the server is running, a number of error messages in the following form will result:

  ```
  Unable to delete file ../image/idm/WEB-INF/lib/*.jar
  ```

  To correct this error, shut down the server and try again.

Out of Memory Errors

If you experience out-of-memory errors as you work with Identity Manager IDE, you might have to increase the NetBeans memory settings. Consult your NetBeans product documentation for instructions.

Tomcat Manager Dialog Displays, Requesting User Name and Password

If you are working with the standard Identity Manager IDE project, start the bundled Tomcat instance, and the Tomcat Manager dialog displays, it generally indicates one of the following conditions:

- Multiple Tomcat instances are running.

  You must insure that only one Tomcat instance is running on the host machine and configured to listen on the same port as the bundled Tomcat.

- A credentials mismatch has occurred.

  The credentials stored as part of the bundled Tomcat server must match those stored on the Server Manager's Username and Password fields. For more information about these field values, go to the following website:

  `http://wiki.netbeans.org/wiki/view/FaqInstallationDefaultTomcatPassword`

To check the port number of the bundled Tomcat and the stored credentials:

1. Select the Identity Manager IDE Runtime tab and expand the Servers and Bundled Tomcat nodes.
2. Right-click the Bundled Tomcat node and select Properties from the pop-up menu.
3. When the Server Manager dialog displays, check the Server Port, Username, and Password field values.

Chapter 2

Working with Rules

This chapter explains how to work with Identity Manager rules and rule libraries. The information is organized into the following sections:

This chapter is organized into the following sections:

NOTE You can use the Sun Identity Manager Integrated Development Environment (Identity Manager IDE) to create, edit, and test rules for your deployment. See Chapter 1, "Using the Identity Manager IDE" for detailed instructions.

You can also use the Identity Manager Business Process Editor (BPE) application to create, edit, and validate rules. See Appendix A, "Using the Business Process Editor."

Understanding Rules and Rule Libraries

This section provides the following information:

What is a Rule?

A *rule* is an object in the Identity Manager repository that contains a function written in the XPRESS, XML Object, or JavaScript languages. Within Identity Manager, rules provide a mechanism for storing frequently used logic or static variables for reuse within forms, workflows, and roles.

You can pass arguments to a rule to control its behavior, and a rule can reference and modify variables maintained by the form or workflow.

NOTE Because the XPRESS and XML Object languages are both written in XML, the XPRESS and XML Object code examples used in this chapter are similar.

This chapter assumes you are familiar with the XPRESS language. For detailed information about using XPRESS, see *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

For information about writing rules in JavaScript, see "Writing Rules in JavaScript" on page 144.

Code Example 2-1 contains a simple XML rule. The rule name is defined by the `name` attribute in the `<Rule>` element, and it returns the string value `john` or `mary`. The rule body is an XPRESS expression within the `<Rule>` element.

Code Example 2-1 Example XML Rule

```
<Rule name='getApprover'>
   <cond><eq><ref>department></ref><s>sales</s></eq>
      <s>john</s>
      <s>mary</s>
   </cond>
</Rule>
```

Why Use Rules?

You can call a rule wherever XPRESS is allowed — most notably in workflows and forms. Rules allow you to encapsulate a fragment of logic or a static value that can then be reused in many locations.

The benefits of saving XPRESS logic or static values for reuse include:

- **Easy maintenance**. You can modify a rule by changing a single object instead of changing all of the forms or workflows that reference the rule.
- **Distributed development**. Users can develop rules that focus on rule requirements without having to be aware of all the forms or workflows that reference that rule.
- **Hiding complexity**. Less technical users can modify simple rules using interfaces that do not require knowledge of XML, workflows, or forms.

Using Rules in Forms

You typically call a rule in forms to calculate the `allowedValues` display property or to control field visibility within a `<Disable>` expression. Within forms, rules could be the most efficient mechanism for storing and reusing:

- A list of corporate departments
- Default values
- A list of office buildings

When calling rules from forms, it is particularly important that you properly secure those forms. For information about securing rules, see *"Securing Rules" on page 153*.

The rule in Code Example 2-2 returns a list of job titles. Rules such as this are often used in Identity Manager forms to calculate lists of names for selection. To add or change a new job title, you only need to modify this rule instead of having to modify all of the forms that reference the rule.

Code Example 2-2 Returning a Job Titles List

```
<Rule name='Job Titles'>
   <list>
      <s>Sales</s>
      <s>Accounting Manager</s>
      <s>Customer Service Representative</s>
   </list>
</Rule>
```

The field in Code Example 2-3 calls the rule defined in the preceding example to use the job titles list in a select box:

NOTE In this example, the `rule name` element uses a lowercase ***r*** because you use it to call the rule, not define it.

Code Example 2-3 Using a Job Titles List in a Select Box

```
<Field name='global.jobTitle'>
   <Display class='Select'>
      <Property name='title' value='Job Title'/>
      <Property name='allowedValues'>
         <rule name='Job Titles'/>
      </Property>
   </Display>
</Field>
```

Using Rules in Roles

You can use rules to set the value of any resource attribute in a role definition. When the rule is evaluated, it can reference any attribute of the user view.

The rule in Code Example 2-4 sets a value for the user's description of a resource, such as NT. When a user is created with a role that has this rule associated with it, the description value will automatically be set according to the rule.

Code Example 2-4 Setting the Value for a User's Resource Description

```
<Rule name='account description'>
    <concat>
      <s>Account for </s>
      <ref>global.firstname</ref>
      <ref>global.lastname</ref>
      <s>.</s>
    </concat>
</Rule>
```

Identity Manager forms and workflows support rules that can dynamically calculate the name of another rule to call. Code Example 2-5 shows a form field that calls a rule that calculates a department code:

Code Example 2-5 Calling a Rule that Calculates a Department Code

```
<Field name='DepartmentCode'>
   <Display class='Text'>
     <Property name='title' value='DepartmentCode'/>
   </Display>
     <Expansion>
        <rule>
          <cond>
            <eq>
               <ref>var1</ref>
               <s>Admin</s>
            </eq>
            <s>AdminRule</s>
            <s>DefaultRule</s>
          </cond>
        </rule>
     </Expansion>
</Field>
```

Workflow activities can also contain subprocesses that contain a rule that dynamically calculates the subprocess name:

Code Example 2-6 Calculating a Subprocess Name Dynamically

```
<Activity id='0' name='activity1'>
   <Variable name='ValueSetByRule'>
     <rule>
       <cond>
          <eq><ref>var2</ref><s>specialCase</s></eq>
          <s>Rule2</s>
          <s>Rule1</s>
       </cond>
       <argument name='arg1'>
          <ref>variable</ref>
       </argument>
     </rule>
   </Variable>
</Activity>
```

Using Rules in Workflows

In workflow, you can use a rule to:

- Calculate an approver
- Add a condition to a transition
- Implement an action
- Calculate an approval escalation timeout

Code Example 2-7 is a manual action used to send an approval request to an administrator. You can specify a timeout value for this action, and if the administrator does not respond within the specified time, the action terminates, and the workflow escalates the approval to a different administrator.

In this example, the timeout is specified with a fixed value of 86,400 seconds or 24 hours.

Code Example 2-7 Specifying a Timeout with a Fixed Value

```
<Rule name='Approval Timeout'>
<i>86400</i>
</Rule>
```

Code Example 2-8 is a manual action that calls the `timeout` rule:

Code Example 2-8 Calling the timeout Rule

```
<ManualAction>
   <Owner name='$(approver)'/>
   <Timeout>
      <rule name='Approval Timeout'/>
   </Timeout>
   <FormRule>
      <ref>approvalForm</ref>
   </FormRule>
</ManualAction>
```

What is a Rule Library?

A rule library is an XML configuration object that is stored in the Identity Manager repository. The configuration object contains a *library* object, which in turn contains one or more *rule* objects.

Creating *rule libraries* is a convenient way to organize closely related rules into a single object. Using libraries can simplify rule maintenance by reducing the number of objects in the repository and makes it easier for form and workflow designers to identify and call useful rules.

For example, Code Example 2-9 shows a library containing two different account ID generation rules:

Code Example 2-9 Using a Rule Library with Two Account ID Generation Rules

```
<Configuration name='Account ID Rules'>
   <Extension>
      <Library>
         <Rule name='First Initial Last'>
            <expression>
               <concat>
                  <substr>
                     <ref>firstname</ref>
                     <i>0</i>
                     <i>1</i>
                  </substr>
                  <ref>lastname</ref>
               </concat>
            </expression>
         </Rule>
         <Rule name='First Dot Last'>
            <expression>
               <concat>
                  <ref>firstname</ref>
                  <s>.</s>
                  <ref>lastname</ref>
               </concat>
            </expression>
         </Rule>
      </Library>
   </Extension>
</Configuration>
```

You can use the Identity Manager IDE to view and edit the default rule libraries or to add new rules to an existing library object. For detailed instructions, see "Working with Repository Objects" on page 41.

Customizing Default Rules and Rule Libraries

You can use the Identity Manager IDE to edit the default Identity Manager rules to follow a custom set of steps. Identity Manager ships with a library of default rules and rule libraries, including:

- Active Sync Rules
- AlphaNumeric Rules Library
- Auditor Rules
- Excluded Resource Accounts Rule SubType
- Naming Rules Library
- RegionalConstants Library

Active Sync Rules

When an Active Sync adapter detects a change to an account on a resource, it either maps the incoming attributes to an Identity Manager user, or it creates an Identity Manager user account.

NOTE Active Sync rules must use `context`, not `display.session`.

Table 2-1 lists the predefined Active Sync rules.

Table 2-1 Predefined Active Sync Rules

Rule Name	Description
ActiveSync has isDeleted set	Used by migration from resources with "Process deletes as updates" set to false.
No Correlation Rule	Default rule to use when no correlation is desired.
No Confirmation Rule	Default rule to use when no confirmation is desired.

AlphaNumeric Rules Library

A default library of alphanumeric rules enables you to control how numbers and letters are ordered and displayed in Identity Manager forms and workflows. In the Identity Manager IDE, this library is displayed as the `Alpha Numeric Rules` library object.

Table 2-2 lists the rules in this library.

Table 2-2 Default Alphanumeric Rules

Rule Name	Description
AlphaCapital	List of English uppercase alphabetic characters
AlphaLower	List of English lowercase alphabetic characters
Numeric	List of numeric characters
WhiteSpace	List of white space characters
SpecialCharacters	List of common special characters
IllegalNTCharacters	List of illegal NT characters
legalEmailCharacters	Tests to see if `str` is all numeric characters.
isNumeric	Tests to see if `str` is only numeric
isAlpha	Tests to see if `str` is only alpha
hasSpecialChar	Tests to see if `str` has any special characters
hasWhiteSpace	Tests to see if `str` has any white space characters
isLegalEmail	Tests to see is `str` contains legal characters for an email address
hasIllegalNTChar	Tests to see is `str` is all numeric characters
stringToChars	Converts the supplied string (passed as a `testStr` argument) to a list of its component characters.
StripNonAlphaNumeric	Removes any nonalphanumeric characters from `testStr`

Auditor Rules

To achieve a high level of configurability with minimal complexity, Identity Auditor makes judicious use of rules in audit policy and access scan object configuration.

Table 2-3 provides an overview of the rules you can use to customize how audit policy remediation works and how access scans operate.

Table 2-3 Auditor Rule Types Quick Reference

Rule Type	Example Rules	subTypes and authTypes	Purpose
Attestor	Default Attestor	**SubType**: `ATTESTORS_RULE` **AuthType**: `AccessScanRule`	Automates the attestation process by specifying a default attestor for manual entitlements.
Attestor Escalation	Default EscalationAttestor	**SubType**: `AttestorEscalationRule` **AuthType**: `AccessScanRule`	Automates the attestation process by specifying a default escalation user for manual attestation.
Audit Policy	Compare Accounts to Roles	**SubType**: `SUBTYPE_AUDIT_POLICY_RULE` **SubType**: `SUBTYPE_AUDIT_POLICY_SOD_RULE` **AuthType**: `AuditPolicyRule`	Compares user accounts to accounts specified by current Roles.
	Compare Roles to Actual Resource Values	**SubType**: `SUBTYPE_AUDIT_POLICY_RULE` **SubType**: `SUBTYPE_AUDIT_POLICY_SOD_RULE` **AuthType**: `AuditPolicyRule`	Compares current resource attributes with those specified by current Roles.
Remediation User Form		**SubType**: `USER_FORM_RULE` **AuthType**: Not specified	Automates the attestation process by allowing audit policy authors to constrain which part of a user view is visible when responding to a particular policy violation.
Remediator	Default Remediator	**SubType**: `REMEDIATORS_RULE` **AuthType**: `AccessScanRule`	Automates the remediation process by specifying a remediator for any entitlements created in remediating state.

Table 2-3 Auditor Rule Types Quick Reference*(Continued)*

Rule Type	Example Rules	subTypes and authTypes	Purpose
Review Determination	Reject Changed User	**SubType**: `REVIEW_REQUIRED_RULE` **AuthType**: `AccessScanRule`	Automates the attestation process by automatically rejecting user entitlement records.
	Review Changed Users	**SubType**: `REVIEW_REQUIRED_RULE` **AuthType**: `AccessScanRule`	Automates the attestation process by automatically approving user entitlement records.
	Review Everyone	**SubType**: `REVIEW_REQUIRED_RULE` **AuthType**: `AccessScanRule`	Automates the attestation process by requiring manual attestation for some user entitlement records.
User Scope	All Administrators	**SubType**: `USER_SCOPE_RULE` **AuthType**: `AccessScanRule`	Provides flexibility in selecting a list of users to be scanned by an access scan.
	All Non-Administrators	**SubType**: `USER_SCOPE_RULE` **AuthType**: `AccessScanRule`	Provides flexibility in selecting a list of users to be scanned by an access scan.
	Users Without a Manager	**SubType**: `USER_SCOPE_RULE` **AuthType**: `AccessScanRule`	Provides flexibility in selecting a list of users to be scanned by an access scan.
ViolationPriority	ViolationPriority	**SubType**: Not specified **AuthType**: `EndUserAuditorRule`	Customization — allows the deployment to specify what are valid violation priorities and the corresponding display strings.
ViolationSeverity	ViolationSeverity	**SubType**: Not specified **AuthType**: `EndUserAuditorRule`	Customization — allows the deployment to specify what are valid violation severities and the corresponding display strings.

The following sections provide information about these Identity Auditor rules, how you might customize them, and why:

- Attestor Rule
- Attestor Escalation Rule
- Audit Policy Rule
- Remediation User Form Rule
- Remediator Rule
- Review Determination Rule

- User Scope Rules
- ViolationPriority Rule
- ViolationSeverity Rule
- Sample Auditor Rule Multiple Account Types

Attestor Rule

Every user entitlement that is created in a pending state must be attested by someone. During an access review, Identity Auditor passes each user view to the Attestor rule to determine who gets the initial attestation requests.

The `idmManager` attribute on the `WSUser` object contains the Identity Manager account name and ID of the user's manager.

- If you define a value for `idmManager`, the Attestor rule returns `idmManager` as the attestor for the user represented by the entitlement record.
- If the `idmManager` value is null, the Attestor rule returns `Configurator` as the attestor.

You can use alternate implementations to designate both `IdmManager` and any Resource owners as attestors (for Resources included in the view). This rule takes the current user view and a `LighthouseContext` object as inputs, so you can use any data known to Identity Manager.

Inputs: Accepts the following input variables:

- `lhcontext`: LighthouseContext
- `userEntitlement`: Current user view

You must specify the following for a custom Attestor rule:

SubType: `ATTESTORS_RULE`

AuthType: `AccessScanRule`

Called: During access scan; after evaluating all audit policies, but before dispatching the user entitlement

Returns: A list of zero or more Identity Manager attestor names (users responsible for attesting a particular user entitlement) or `NamedValue` pairs.

- If the result is a string, it must resolve to an Identity Manager account ID. If delegation is enabled for the access scan, the access scan will use the delegation settings of the Identity Manager user returned by the code.
- If the result is a `NamedValue`, it assumed to be a bound delegation pair `[Delegator, Delegatee]`, and the access scan will not resolve any further.

> **NOTE** No checking is performed on `NamedValue` pairs. If the rule returns `NamedValue` pair elements, they are passed on without validation.

- If the result is not a valid Identity Manager user name, the rule appends errors to the scan task results, but the scan thread continues.
- If the result is a zero-length list, the attestation request remains in pending state because nobody will process the request.
- If the result is neither a string or a `NamedValue`, an exception results and the scan thread aborts.

Predefined Rule: Default Attestor

Location: Compliance > Manage Policies > Access Scan > Attestor Rule

Attestor Escalation Rule

A workflow calls the Attestor Escalation rule when an attestation times out because the attestor did not take action within a specified period of time. This rule returns the next person in the *escalation chain* based on the cycle count.

Inputs: Accepts the following input variables:

- `wfcontext`: WorkflowContext
- `userEntitlement`: View of user entitlement, including user view
- `cycle`: Escalation level. For the first escalation, the cycle is 1.
- `attestor`: Name of attestor who failed to attest before the attestation request timed out.

You must specify the following for a custom Attestor Escalation rule:

SubType: `AttestorEscalationRule`

AuthType: `AccessScanRule`

Called: During an attestation workflow when a workitem times out. (Default timeout is *0* — never times out).

Returns: A single attestor name or a list of attestor names, which must be valid Identity Manager account names.

- If the attestor does not have a manager, the Attestor Escalation rule returns `Configurator`.
- If the result is an invalid account name or null, the attestation workitem is not escalated.

Predefined Rule: Default EscalationAttestor

Location: Compliance > Manage Policies > Access Scan > Attestor Escalation Rule

Audit Policy Rule

An audit policy contains a set of rules that it applies to data representing an object being audited. Each rule can return a boolean value (plus some optional information).

To determine whether a policy has been violated, the audit policy evaluates a logical operation on the results of each rule. If the audit policy has been violated, a compliance violation object might result, with (typically) one compliance violation object per policy, rule, or whatever was being audited. For example, an audit policy with five rules might result in five violations.

Inputs: None

You must specify the following for a custom Audit Policy rule:

SubTypes:

- `SUBTYPE_AUDIT_POLICY_RULE` (for an audit policy rule)
- `SUBTYPE_AUDIT_POLICY_SOD_RULE` (for an audit policy SOD rule)

SOD (*separation* or *segregation of duties*) rules differ from regular rules in that they are expected to produce a list element in the rule output. A list element is not *required*; but if one is not present, it causes any corresponding violations to be ignored in SOD reporting.

AuthType: `AuditPolicyRule`

NOTE When you use the Audit Policy Wizard to create an Audit Policy rule, the wizard uses the `AuditPolicyRule` authType by default.

If you use the Identity Manager IDE or the Identity Manager Business Process Editor (BPE) to create an Audit Policy rule, be sure to specify the `AuditPolicyRule` authType.

Called: During an Audit Policy Evaluation

Returns: An audit policy rule must return an integer value, but the value can be expressed as one of the following:

- A pure integer:

```
<i>1</i>
```

- An integer within a map of additional data:

```
<map>
  <s>result</s>
  <i>1</i>
  ...
</map>
```

If the audit policy returns a map, other elements can affect the resulting compliance violation. These elements include:

- **`resources` element**: Causes the compliance violation to refer to two resources, `resource one` and `resource two`. These values must be real resource names because the compliance violation contains actual object references (so the names are resolved to IDs). (Default is *no resource*.)

```
<s>resources</s>
<list>
    <s>resource one</s>
    <s>resource two</s>
</list>
```

- **`severity` element**: Causes the compliance violation to have the specified severity. (Default is *1*.)

```
<s>severity</s>
<i>3</i>
```

- **`priority` element**: Causes the compliance violation to have the specified priority. (Default is *1*.)

```
<s>priority</s>
<i>2</i>
```

- **`violation` element**: Prevents the audit scanner from creating a rule violation — even if the audit policy evaluates to `true`.

  By default, if the audit policy evaluates to `true`, it creates compliance violations for all rules that return non-zero. Setting this element to zero allows the rule to return `true`, but not have a violation created for it.

```
<s>violation</s>
<i>0</i>
```

NOTE The Audit Policy Wizard only creates rules that reference a single resource and return an integer value (not a map).

To use any of the preceding map-related features, you must write the rule yourself. Some very sophisticated audit policy rule examples are provided in `sample/auditordemo.xml`.

Predefined Rules:

- **Compare Accounts to Roles**: Compares user accounts to accounts specified by roles. Any account not referenced by a role is considered an error.
- **Compare Roles to Actual Resource Values**: Compares current resource attributes with those specified by current Roles. Any differences are considered errors, and any resources or resource attributes not specified by a role are ignored.

Location: None

Remediation User Form Rule

The Remediation User Form rule allows audit policy authors to constrain which part of a user view is visible when they are responding to a particular policy violation.

When a remediator edits a user during entitlement remediation processing, a JSP (`approval/remModifyUser.jsp`) calls the Remediation User Form rule. This rule allows the access scan to specify an appropriate form for editing a user. If the remediator has already specified a user form, then the access scan uses that form instead.

Inputs: Accepts the `item` variable (Remediation WorkItem)

You must specify the following for a custom Remediation User Form rule:

Subtype: `USER_FORM_RULE`

AuthType: Not specified

Called: During JSP form processing after the remediator clicks Edit User on the remediation form.

Returns: The name of a User Form or a null.

Predefined Rules: None

Locations:

- Compliance > Manage Policies > Access Scan > Remediation User Form Rule
- Compliance > Manage Policies > Audit Policy > Remediation User Form Rule

Remediator Rule

During an access review, every user view is passed to the Remediator rule to determine who should get the initial remediation requests. This rule is analogous to the Attestors rule, except the Remediator rule is called when a workitem is created in the remediating state.

Inputs: Accepts the following input variables:

- `lhcontext`: LighthouseContext
- `userEntitlement`: Current user view

You must specify the following for a custom Remediator rule:

SubType: `REMEDIATORS_RULE`

AuthType: `AccessScanRule`

Called: During access scan, after evaluating all audit policies and before dispatching the user entitlement

Returns: A list of zero or more Identity Manager remediator names or `NamedValue` pairs.

- If the result is a string, it is resolved to a Identity Manager user, and if delegation is enabled for the access scan, the user's delegation data is used.
- If the result is a `NamedValue`, it is assumed to be a bound delegation pair [**Delegator**, `Delegatee`].
- If the result is one or more invalid Identity Manager user names, errors indicating a problem are appended to the scan task results, but the scan thread continues.
- If the result is not a string or `NamedValue`, an exception occurs and the scan thread aborts.
- If the results are a zero-length list, the remediation request remains in a pending state because nobody will process it.

NOTE No checking is performed on `NamedValue` pairs. If the rule returns `NamedValue` pair elements, they are passed on without validation.

Predefined Rule: Default Remediator

Location: Compliance > Manage Policies > Access Scan > Remediator Rule

Review Determination Rule

During an access review, every user view is passed to the Review Determination rule to determine whether the corresponding user entitlement record can be automatically approved or rejected, automatically placed into remediation state, or if it must be manually attested.

You can use the Review Determination rule to significantly increase the efficiency of an access review.

If you can encapsulate any institutional knowledge that would allow a user to be automatically approved or rejected, and express that knowledge in this rule, you reduce the number of manual attestations needed and improve overall review performance.

As a further improvement, this rule can return information that will be visible to the attestors as a "hint." For example, if the rule noticed that the user has privilege access to a resource, see `attestorHint` in Example 1 on page 125.

In addition, the Review Determination rule can access the user view (including any Compliance Violations) and compare the user's previous UserEntitlements, which allows the rule to approve or reject all UserEntitlements that are the same as (or different from) a previously approved UserEntitlement.

Inputs: Accepts the following input variables:

- `context`: LighthouseContext
- `review.scanId`: Current access scan ID
- `review.username`: Identity Manager account name of user being scanned
- `review.userId`: Identity Manager ID of user being scanned
- `attestors`: Attestors' Identity Manager account names
- `userView`: Current user view

You must specify the following for a custom Review Determination rule:

SubType: `REVIEW_REQUIRED_RULE`

AuthType: `AccessScanRule`

Called: During access scan, after evaluating all audit policies and before dispatching the user entitlement

Returns: An integer or a map

- If the rule returns an integer, its value is interpreted as follows:
  - **-1**: No attestation required
  - **0**: Automatically reject attestation
  - **1**: Manual attestation
  - **2**: Automatically approve attestation
  - **3**: Automatically remediate attestation

    When the attestation is set to auto-remediating mode, Identity Manager creates an `AccessReviewRemediation` work item and routes the work item through the Remediator rule associated with the access scan.
- If the rule returns a map, the output must be similar to one of the following examples:

  Example 1: Manually attests the UserEntitlement. The rule provides a hint to the manual attestor.

```
<map>
   <result>
   <i>1</i>
   <s>reason</s>
   <s><reason that the attestation was auto-approved/rejected></s>
   <s>attestorHint</s>
   <s><hint to attestor></s>
</map>
```

NOTE The `attestorHint` value in the output map *must* be a string or a list of strings.

Example 2: Automatically rejects the UserEntitlement. The rejection comment indicates that group membership is disallowed.

```
<map>
  <s>result</s>
  <i>0</i>
  <s>reason</s>
  <s>User belongs to NT group Domain Administrators</s>
</map>
```

NOTE The value of `attestorHint` is shown to the attestor through the user interface. The value of `reason` is recorded in the attestation history.

Predefined Rules:

- **Reject Changed Users**: Automatically rejects user entitlements that have changed since the last approval state, and automatically approves user entitlements that are unchanged. All unknown UserViews are forwarded for manual attestation. Only the `accounts` section of the User view is compared.
- **Review Changed Users**: Automatically approves any users whose account data has not changed since their last approved entitlement. Users with changed account data or no approved data are manually attested. Only the `accounts` section of the User view is compared.
- **Review Everyone**: Forwards all user entitlement records for manual attestation.

Location: Compliance > Manage Access Scans > Access Scan > Review Determination Rule

User Scope Rules

If an access scan has users scoped by a rule, the User Scope rule is evaluated to determine a list of users to scan.

Inputs: Accepts the `lhcontext` variable (LighthouseContext)

You must specify the following for a custom User Scope rule:

SubType: `USER_SCOPE_RULE`

AuthType: `AccessScanRule`

Called: At the beginning of an access scan

Returns: An Identity Manager user name or a list of Identity Manager user names. Each name must be a valid Identity Manager user name.

- If the results contain any names that cannot be resolved to valid Identity Manager user names, the rule returns an error.
- If the results contain any duplicate user names, the rule returns an error.

NOTE

- An access scan that scans the same user multiple times might fail to create the attestation workflow for a subsequent instance of the same user. Therefore, a customized implementation of the User Scope rule should provide checks to avoid duplicate users in the output.
- This rule can return accounts that are not available to the administrator running the scan. In this case, the scan will attempt to get the account's user view and fail; resulting in an error in the scan task.

Predefined Rules:

- **All Administrators**: Returns all users with administrative capabilities assigned.
- **All Non-Administrators**: Returns all users with no administrative capabilities assigned.
- **Users Without Manager**: Returns all user accounts with no manager (`idmManager`) assigned.

Location: Compliance > Manage Access Scans > Access Scan > User Scope Rule

ViolationPriority Rule

Use the ViolationPriority rule to allow a deployment to specify what the valid violation priorities are, and what the corresponding display strings will be.

Inputs: None

You must specify the following for a custom ViolationPriority rule:

SubType: Not specified

AuthType: `EndUserAuditorRule`

Called: When displaying the violation list and when changing violation priority.

Returns: A list of `key/value` pairs indicating priority integer value and a corresponding string. The integer values must be contiguous because the rule returns a list, not a map.

NOTE You can customize this rule to change the display value for any priority setting.

When a ComplianceViolation is created, you can change priority values in the Remediation WorkItem list viewer. Select one or more Remediation WorkItems, and then select Prioritize, which enables you to change priority values.

To see these values in the Remediation WorkItem list view, you must change the `approval/remediate.jsp` page by setting the `includeCV` option to **`true`** (default is `false`). However, enabling the more detailed view affects performance, which may be unacceptable for deployments with lots of Remediations.

The custom value expects the ViolationPriority rule to be an array rather than a map. So, if you use 100 as the integer value, the rule must have 200 elements (alternate int/string). The list provides both string mapping for the integer *and* populates the selection in the form where you changed it.

Predefined Rule: ViolationPriority

Location: Called from the Remediation List Form

ViolationSeverity Rule

Use the ViolationSeverity rule to allow a deployment to specify what the valid violation severities are, and what the corresponding display strings will be.

Inputs: None

You must specify the following for a custom ViolationSeverity rule:

SubType: Not specified

AuthType: `EndUserAuditorRule`

Called: When displaying the violation list and when changing violation severity.

Returns: A list of `key/value` pairs indicating severity integer value and a corresponding string. The integer values must be contiguous because the rule returns a list, not a map.

NOTE You can customize this rule to change the display value for any priority setting.

When a ComplianceViolation is created, you can change severity values in the Remediation WorkItem list viewer. Select one or more Remediation WorkItems, and then select Priority, which enables you to change severity values.

To see these values in the Remediation WorkItem list view, you must change the `approval/remediate.jsp` page by setting the `includeCV` option to **`true`** (default is `false`). However, enabling the more detailed view affects performance, which may be unacceptable for deployments with lots of Remediations.

The custom value expects the ViolationSeverity rule to be an array rather than a map. So, if you use 100 as the integer value, the rule must have 200 elements (alternate int/string). The list provides both string mapping for the integer *and* populates the selection in the form where you changed it.

Predefined Rule: ViolationSeverity

Location: Called from the Remediation List Form

Sample Auditor Rule Multiple Account Types

You can use the Sample Auditor Rule Multiple Account Types rule to dynamically test multiple user accounts per resource. For example,

1. Set up a resource with multiple account types (see Code Example 2-10).

Code Example 2-10 Sample Auditor Rule Multiple Account Types Rule

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Waveset PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Waveset>
<Rule subtype='IdentityRule' name='Administrator Identity'>
  <concat>
     <s>adm</s>
     <ref>attributes.accountId</ref>
  </concat>
</Rule>
</Waveset>
```

2. Add a user with two accounts on the resource and set up a user form so that the new resource attributes are directly assigned separately:

   `account[Simulated Resource].department`

   `account[Simulated Resource|admin].department`

3. Assign different values for each account and test the policy rule.

Location: `sample/rules/SampleAuditorRuleMultipleAccountTypes.xml`

Compliance Violation Rules

ComplianceViolations support numeric severity and priority attributes that enable you to distinguish between violations by severity or priority. You can assign these attributes to the violation, based on Audit rule output.

For example, if the Audit rule provides the following output, the resulting ComplianceViolation will have a severity of `3` and a priority of `4`.

```
<map>
  <s>result</s>
  <i>1</i>
  <s>severity</s>
  <i>3</i>
  <s>priority</s>
  <i>4</i>
</map>
```

The following rules map between a ComplianceViolation's numeric value and its display string value:

- **ViolationSeverity**: Indicates how serious the violation is.
- **ViolationPriority**: Indicates the order in which a ComplianceViolation should be addressed.

Identity Auditor allows you to customize these rules by changing the display value for any severity or priority setting.

After a ComplianceViolation is created, you can view and change severity and priority values in the Remediation WorkItem list viewer by selecting one or more Remediation WorkItems, and then clicking Prioritize.

NOTE To view severity and priority values in the Remediation WorkItem list viewer, you must change the `approval/remediate.jsp` page to set the `includeCV` option to `true` (default is `false`).

However, be aware that enabling the more-detailed view affects performance, which may be unacceptable for deployments with lots of Remediations.

DateLibrary

The DateLibrary a default library of rules that control how dates and times are displayed in a deployment. In the Identity Manager IDE, this library is displayed as the `Date Library` library object.

Use these rules to customize how dates and times are displayed:

- `Date Validation`: Determines valid date strings.

  This rule takes one argument in the form `mm/dd/yy`. (If month or day values are provided in with single digits, the rule accounts for them appropriately.)

  Returns: A `true` if the string provided contains valid date components

- `Validate Day Month Year`: Determines valid day, month, and year strings.

  This rule takes three arguments `month`, `day`, and `year`. If the month or the day values are provided in with single digits, the rule accounts for them appropriately.

  Returns: A `true` if the string provided is a valid date

- `Validate Time`: Determines valid time strings.

  This rule takes one argument in the form `HH:mm:ss`. If the time string is not in this format, or the components are out of bounds (for example, if the hour is less than zero or greater than 23) the rule will return `false`.

  Returns: A `true` if the string provided is a valid time

Excluded Resource Accounts Rule SubType

The `ExcludedAccountRule` subType supports the exclusion of resource accounts from resource operations.

Inputs: Accepts the following input variables:

- `accountID`: String account ID being tested.

  You can compare the `accountID` parameter to one or more resource accounts that should be excluded from Identity Manager.

- `operation`: Resource operation to be performed.

  The rule can use the `operation` parameter to have finer control over which resource accounts are exempt from the actions specified by the `operation` parameter. If an `operation` parameter is not used within the rule, all accounts identified by the rule will be excluded from all the listed operations.

  The `operation` parameter can contain the following values:

  - `create`
  - `update`
  - `delete`
  - `rename` (used when the only detected change is a new account ID)
  - `rename_with_update`
  - `list`
  - `iapi_create` (only used within Active Sync)
  - `iapi_update` (only used within Active Sync)
  - `iapi_delete` (only used within Active Sync)

authType: ExcludedAccountsRule

Code Example 2-11 exemplifies subType use, and it excludes specified resource accounts for UNIX adapters.

Code Example 2-11 Exemplifying Subtype Use

```
<Rule name='Excluded Resource Accounts' authType='ExcludedAccountsRule'>
   <RuleArgument name='accountID'/>
   <defvar name 'excludedList'>
      <List>
         <String>root</String>
         <String>daemon</String>
         <String>bin</String>
         <String>sys</String>
         <String>adm</String>
         <String>uucp</String>
         <String>nuucp</String>
         <String>listen</String>
         <String>lp</String>
      </List>
   </defvar>
   <cond>
      <eq>
         <contains>
            <ref>excludedList</ref>
            <ref>accountID</ref>
         </contains>
         <i>1</i>
      </eq>
      <Boolean>true</Boolean>
      <Boolean>false</Boolean>
   </cond>
</Rule>
```

Code Example 2-12 illustrates the use of the `operation` parameter. This parameter allows the "Test User" resource account to be manipulated — without impacting Identity Manager — if Active Sync is running against the resource.

Code Example 2-12 Using the operation Parameter

```
<Rule name='Example Excluded Resource Accounts'
authType='ExcludedAccountsRule'>
```

Code Example 2-12 Using the operation Parameter *(Continued)*

```
<!--
Exclude all operations on 'Administrator' account
Exclude activeSync events on 'Test User' account
-->

   <RuleArgument name='accountID'/>
   <RuleArgument name='operation'/>

<!-- List of IAPI Operations -->
   <defvar name='iapiOperations'>
      <List>
         <String>iapi_create</String>
         <String>iapi_update</String>
         <String>iapi_delete</String>
      </List>
   </defvar>
   <or>

   <!-- Always ignore the administrator account. -->
      <cond>
         <eq>
            <s>Administrator</s>
            <ref>accountID</ref>
         </eq>
         <Boolean>true</Boolean>
         <Boolean>false</Boolean>
      </cond>

<!-- Ignore IAPI events for the 'Test User' account -->
      <and>
         <cond>
            <eq>
               <contains>
                  <ref>iapiOperations</ref>
                  <ref>operation</ref>
               </contains>
               <i>1</i>
```

Code Example 2-12 Using the operation Parameter *(Continued)*

```
            </eq>
            <Boolean>true</Boolean>
            <Boolean>false</Boolean>
         </cond>
         <cond>
            <eq>
               <ref>accountID</ref>
               <s>Test User</s>
            </eq>
            <Boolean>true</Boolean>
            <Boolean>false</Boolean>
         </cond>
      </and>
   </or>
</Rule>
```

Naming Rules Library

The Naming Rules Library is a default library of naming rules that enable you to control how names are displayed after rule processing. In the Identity Manager IDE, this library is displayed as the `NamingRules` library object.

Table 2-4 lists the default naming rules.

Table 2-4 Default Naming Rules

Rule Name	Description/Output
AccountName — First dot Last	Marcus.Aurelius
AccountName — First initial Last	MAurelius
AccountName — First underscore Last	Marcus_Aurelius
Email	marcus.aurelius@example.com
Fullname — First space Last	Marcus Aurelius
Fullname — First space MI space Last	Marcus A Aurelius
Fullname — Last comma First	Aurelius, Marcus

RegionalConstants Library

The RegionalConstants Library is a default library of regional constants rules that enable you to control how states, days, months, countries, and provinces are displayed. In the Identity Manager IDE, this library is displayed as the `RegionalConstants Rules` library object.

Table 2-5 lists the default regional constants rules.

Table 2-5 Default Regional Constants Rules

Rule Name	Description
US States	List of the full names of the US states
US State Abbreviations	List of the standard US state abbreviations.
Days of the Week	List of the full names of the seven days of the week.
Work Days	List of the five work days of the week (U.S.)
Months of the Year	List of the full names of the months of the year.
Month Abbreviations	List of the standard abbreviation for the selected month.
Numeric Months of the Year	Returns a list of 12 months.
Days of the Month	Returns a list of 31 days.
Smart Days of the Month	Returns a list based on a numeric month and four-digit year.
Countries	Lists the names, in English, of the countries of the world.
Canadian Provinces	Lists the names, in English, of the Canadian provinces.

Developing New Rules and Rule Libraries

This section describes how to develop rules for your deployment, and provides the following information:

- Understanding Rule Syntax
- Writing Rules in JavaScript

NOTE

- For information about applying rules to a roles, see "Using Rules in Roles" on page 108 and *Sun Java™ System Identity Manager Administration*.
- For information about adding rules to an existing rule library, see "Customizing Default Rules and Rule Libraries" on page 112.
- For information about using XPRESS to write an expression, see the XPRESS Language chapter in *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

TIP

When designing a rule, try to maximize the ease with which a less experienced user could further customize the rule using the Identity Manager IDE.

A complex rule, with well chosen rule arguments, can be extensively customized by changing default values, without ever having to expose XPRESS or JavaScript to the user.

Understanding Rule Syntax

Rules are typically written in XML and encapsulated in the `<Rule>` element.

This section covers the following topics:

- Using the <Rule> Element
- Returning Fixed Values
- Referencing Variables
- Declaring a Rule with Arguments
- Rules with Side Effects

Using the <Rule> Element

Code Example 2-13 shows the use of the `<Rule>` element to define a basic rule expression. The `name` property identifies the name of the rule. The rule is written in XPRESS.

Code Example 2-13 Using the <Rule> Element to Define a Basic Rule Expression

```
<Rule name='getApprover'>
   <cond><eq><ref>department></ref><s>sales</s></eq>
      <s>Sales Manager</s>
      <s>HR Manager</s>
   </cond>
</Rule>
```

Returning Fixed Values

If the rule returns a fixed value, you can write it using XML Object syntax. Code Example 2-14 returns a list of strings.

Code Example 2-14 Returning a List of Strings

```
<Rule name='UnixHostList'>
   <List>
      <String>aas</String>
      <String>ablox</String>
      <String>aboupdt</String>
   </List>
</Rule>
```

NOTE For more information about XML Object syntax, see the XML Object Language chapter in *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

Referencing Variables

Rules are allowed to reference the value of external variables with a `<ref>` expression. The names of the available variables are determined by the context in which the rule is used. When used within forms, any form field, view attribute, or variable defined with `<defvar>` can be referenced. When used within workflow, any variable defined within the workflow process can be referenced.

In Code Example 2-15, a form uses a rule to calculate an email address. The form defines the `global.firstname` and `global.lastname` fields, and the rule references those fields. The email address is calculated by concatenating the first letter of `global.firstname` with `global.lastname` and the `@example.com` string.

Code Example 2-15 Calculating an Email Address

```
<Rule name='Build Email'>
   <concat>
      <substr> <ref>global.firstname</ref> <i>0</i> <i>1</i> </substr>
      <ref>global.lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

In Code Example 2-16, a workflow uses a rule to test whether a transition to a particular activity should be taken. The workflow defines a variable named `user` that contains the User View. The rule returns true if this user has any NT resources assigned to them or null if no NT resources are assigned. The workflow engine interprets null as false and would consequently not take the transition.

Code Example 2-16 Testing a Transition

```
<Rule name='Has NT Resources'>
   <notnull>
      <ref>user.accountInfo.types[NT].accounts</ref>
   </notnull>
</Rule>
```

Declaring a Rule with Arguments

Though rules are not required to declare arguments, it is considered good practice to do so. Declaring arguments provides documentation to the rule author, allows reference validation in the Identity Manager IDE, and allows the rule to be used in forms and workflows that may not use the same naming convention.

To declare rule arguments, use the `<RuleArgument>` element. The argument can have a default value set by specifying a value after the argument name as shown in the following location argument.

Code Example 2-17 Setting a Default Value

```
<Rule name='description'>
   <RuleArgument name='UserId'/>
   <RuleArgument name='location' value='Austin'/>
   <concat>
      <ref>UserId</ref>
      <s>@</s>
      <ref>location</ref>
   </concat>
</Rule>
```

NOTE When defining a rule, use the `<Rule>` element that has an uppercase **R** as in `<Rule name='rulename'>`. When calling a rule, the XPRESS `<rule>` element has a lowercase **r**, as in `<rulename='rulename'>`.

You can use this rule in a user form, but `UserId` and `location` are not attributes of the user view. To pass the expected arguments into the rule the `<argument>` element is used in the rule call. Note that passing an argument whose name is `location` will override the default value declared in the `RuleArgument` element.

Code Example 2-18 Overriding a Default Value Declared in RuleArgument

```
<rule name='description'>
   <argument name='UserId' value='$(waveset.accountId)'/>
   <argument name='location' value='global.location'/>
</rule>
```

For more information about calling rules, see "Referencing Rules" on page 144.

There is no formal way to declare argument type, but you can specify type in a comment field. Use the <Comment> element to include comments in your rule:

Code Example 2-19 Using <Comment> to Include Comments in a Rule

```
<Comments>
Description rule is expecting 2 arguments. A string value
UserId, which is the employees' ID number, and a string
value location that describes the building location for
the employee
</Comments>
```

TIP If you are using the Identity Manager IDE to edit rules, you might find it helpful to formally define a list of rule arguments. This list would consist of the names of variables that are expected to be available to the rule. You can use them afterwards to perform validation in the Identity Manager IDE.

Rules with Side Effects

Rules typically return a single value, but in some cases you may want a rule to return several values. While a rule expression can return only a single value, a rule can assign values to external variables with the following XPRESS expressions:

- <setvar>
- <setlist>
- <putmap>

In Code Example 2-20, the rule tests the value of the external variable named department and assigns values to two other variables.

Code Example 2-20 Testing the department Variable and Assigning Other Variables

```
<Rule name='Check Department'>
    <switch>
      <ref>global.department</ref>
      <case>
        <s>Engineering</s>
        <block>
          <setvar name='global.location'>
            <s>Building 1</s>
          </setvar>

          <setvar name='global.mailServer'>
            <s>mailserver.somecompany.com</s>
          </setvar>
        </block>
      </case>
      <case>
        <s>Marketing</s>
        <block>
          <setvar name='global.location'>
            <s>Building 2</s>
          </setvar>          <setvar name='global.mailServer'>
            <s>mailserver2.somecompany.com</s>
          </setvar>
        </block>
      </case>
    </switch>
</Rule>
```

In the preceding example, the variables `global.location` and `global.mailServer` are both set according to the value of the variable department. In this case, the return value of the rule is ignored, and the rule is called only for its side effects.

Writing Rules in JavaScript

When rules become complex, you may find it more convenient to write those rules in JavaScript rather than XPRESS. You can wrap the JavaScript in a `<script>` element. For example,

Code Example 2-21 Wrapping JavaScript in a <script> Element

```
<Rule name='Build Email'>
   <script>
      var firstname = env.get('firstname');
      var lastname = env.get('lastname');
      var email = firstname.substring(0, 1) + lastname + "@example.com";
      email;
   </script>
</Rule>
```

To reference the values of form and workflow variables, call the `env.get` function and pass the variable name. You can use the `env.put` function to assign variable names. The value of the last statement in the script becomes the value of the rule. In the preceding example, the rule will return the value in the `email` variable.

You can call other rules with the `env.call` function.

Referencing Rules

You reference rules in a library using an XPRESS `<rule>` expression. The value of the name attribute is formed by combining the name of the configuration object containing the library, followed by a colon, followed by the name of a rule within the library. Therefore, all rule names in a library must be unique.

For example, the following expression calls the rule named `First Dot Last` contained in a library named `Account ID Rules`:

```
<rule name='Account ID Rules:First Dot Last'/>
```

This section provides information about referencing rules. The information is organized as follows:

- Basic Rule Call Syntax
- Rule Argument Resolution

Basic Rule Call Syntax

Rules can be called from anywhere XPRESS is allowed, which includes forms, workflows, or even another rule.

Use the XPRESS `<rule>` expression to call a rule. For example:

```
<rule name='Build Email'/>
```

When the XPRESS interpreter evaluates this expression, it assumes the value of the `name` attribute is the name of a rule object in the repository. The rule is automatically loaded from the repository and evaluated. The value returned by the rule becomes the result of the `<rule>` expression.

In the previous example, no arguments are passed explicitly to the rule. The next example shows an argument passed to the rule using the `argument` element.

```
<rule name='getEmployeeId'>
   <argument name='accountId' value='jsmith'/>
</rule>
```

In the previous example, the value of the argument is specified as a static string `jsmith`. You can also calculate the value of an argument using an expression.

```
<rule name='getEmployeeId'>
   <argument name='accountId'>
      <ref>user.waveset.accountId</ref>
   </argument>
</rule>
```

In the previous example, the argument value is calculated by evaluating a simple `ref` expression that returns the value of the view attribute `user.waveset.accountId`.

Because calculating argument values by referencing attributes is so common, an alternate syntax is also provided.

```
<rule name='getEmployeeId'>
   <argument name='accountId' value='$(user.waveset.accountId)'/>
</rule>
```

The previous examples have the same behavior. They both pass the value of the view attribute `user.waveset.account` as the value of the argument.

Rule Argument Resolution

Most rules contain XPRESS <ref> expressions or JavaScript env.get calls to retrieve the value of a variable. Several options are available for controlling how the values of these variables are obtained.

In the simplest case, the application calling the rule will attempt to resolve all references. For rules called from workflows, the workflow processor will assume all references are to workflow variables. For rules called from forms, the form processor will assume all references are to attributes in a view. Rules can also call another rule by dynamically resolving the called rule's name.

You can also use the optional <RuleArgument> element, which is described in "Declaring a Rule with Arguments" on page 141.

This section provides the following information:

- Calling Scope or Explicit Arguments
- Local Scope Option
- Rule Argument Declarations
- Locked Arguments

Calling Scope or Explicit Arguments

This section provides examples to illustrate rule argument resolution.

Code Example 2-22 illustrates adding a rule to a form that can be used with the User view because there are attributes names in the view:

Code Example 2-22 Adding a Rule to a Form

```
<Rule name='generateEmail'>
   <concat>
      <ref>global.firstname</ref>
      <s>.</s>
      <ref>global.lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

This rule references two variables:

- `global.firstname`
- `global.lastname`.

You can call this rule in a `Field`, as shown in Code Example 2-23:

Code Example 2-23 Calling the Rule in a Field

```
<Field name='global.email'>
   <Expansion>
      <rule name='generateEmail'/>
   </Expansion>
</Field>
```

This method can be a convenient way to write simple rules that are used in user forms only — similar to the concept of *global variables* in a programming language. But there are two problems with this style of rule design. First, it is unclear to the form designer which variables the rule will be referencing. Second, the rule can be called only from user forms because it references attributes of the user view. The rule cannot be called from most workflows because workflows usually do not define variables named `global.firstname` and `global.lastname`.

You can address these problems by passing rule arguments explicitly, and by writing the rule to use names that are not dependent on any particular view.

Code Example 2-24 shows a modified version of the rule that references the variables `firstname` and `lastname`:

Code Example 2-24 Rule Referencing firstname and lastname Variables

```
<Rule name='generateEmail'>
   <concat> \
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

The rule shown in Code Example 2-25 is simpler and more general because it does not assume that the rule will be called from a user form. But the rule must then be called with explicit arguments such as:

Code Example 2-25 Calling the Rule with Explicit Arguments

```
<Field name='global.email'>
   <Expansion>
      <rule name='generateEmail'>
         <argument name='firstname' value='$(global.firstname)'/>
         <argument name='lastname' value='$(global.lastname)'/>
      </rule>
   </Expansion>
</Field>
```

The `name` attribute of the `argument` elements correspond to the variables referenced in the rule. The values for these arguments are assigned to values of global attributes in the user view. This keeps the rule isolated from the naming conventions used by the calling application, and makes the rule usable in other contexts.

Local Scope Option

Even when arguments are passed explicitly to a rule, the system by default allows references to other variables not passed as explicit arguments. Code Example 2-26 shows a workflow action calling the rule but only passing one argument:

Code Example 2-26 Workflow Action Calling the Rule and Passing a Single Argument

```
<Action>
   <expression>
      <setvar name='email'>
         <rule name='generateEmail'>
            <argument name='firstname' value='$(employeeFirstname)'/>
         </rule>
      </setvar>
   </expression>
</Action>
```

When the rule is evaluated, the workflow processor will be asked to supply a value for the variable `lastname`. Even if there is a workflow variable with this name, it may not have been intended to be used with this rule. To prevent unintended variable references, rules should be defined with the `localScope` option.

You enable this option by setting the `localScope` attribute to **`true`** in the `Rule` element:

Code Example 2-27 Setting localScope Attribute to true in a Rule Element

```
<Rule name='generateEmail' localScope='true'>
   <concat>
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

By setting this option, the rule is only allowed to reference values that were passed explicitly as arguments in the call. When called from the previous workflow action example, the reference to the `lastname` variable would return null.

Rules intended for general use in a variety of contexts should always use the local scope option.

Rule Argument Declarations

Though not required, it is considered good practice to include within the rule definition explicit declarations for all arguments that can be referenced by the rule. Argument declarations offer advantages, and can:

- Serve as documentation for the caller of the rule
- Define default values
- Can be used by the Identity Manager IDE to check for misspelled references within the rule
- Can be used by the Identity Manager IDE to simplify the configuration of a rule call

You could rewrite the `generateEmail` rule as shown in Code Example 2-28:

Code Example 2-28 Rewriting the generateEmail Rule

```
<Rule name='generateEmail' localScope='true'>
   <RuleArgument name='firstname'>
      <Comments>The first name of a user</Comments>
   </RuleArgument>
   <RuleArgument name='lastname'>
      <Comments>The last name of a user</Comments>
   </RuleArgument>
   <RuleArgument name='domain' value='example.com'>
      <Comments>The corporate domain name</Comments>
   </RuleArgument>
   <concat>
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@</s>
      <ref>domain</ref>
   </concat>
</Rule>
```

The `Comments` element can contain any amount of text that might be useful to someone examining the rule.

The rule has been modified to define another argument named `domain`, which is given a default value of `example.com`. The default value is used by the rule unless the caller passes an explicit argument named `domain`.

The call in Code Example 2-29 produces the string `john.smith@example.com`:

Code Example 2-29 Producing john.smith@example.com String

```
<rule name='generateEmail'>
   <argument name='firstname' value='john'/>
   <argument name='lastname' value='smith'/>
</rule>
```

The call in Code Example 2-30 produces the string `john.smith@yourcompany.com`:

Code Example 2-30 Producing john.smith@yourcompany.com String

```
<rule name='generateEmail'>
   <argument name='firstname' value='john'/>
   <argument name='lastname' value='smith'/>
   <argument name='domain' value='yourcompany.com'/>
</rule>
```

The call in Code Example 2-31 produces the string `john.smith@`:

Code Example 2-31 Producing john.smith@ String

```
<rule name='generateEmail'>
   <argument name='firstname' value='john'/>
   <argument name='lastname' value='smith'/>
   <argument name='domain'/>
</rule>
```

NOTE In the previous example, a null value is passed for the domain argument, but the default value is not used. If you specify an explicit argument in the call, that value is used even if it is null.

Locked Arguments

Declaring arguments with default values can be a useful technique for making the development and customization of rules easier. If you have a constant value in a rule that may occasionally change, it is easier to locate and change if the value is defined in an argument rather than embedded deep within the rule expression.

The Identity Manager IDE provides a simplified GUI for configuring rules by changing the default values of arguments which is much easier than editing the entire rule expression.

But once an argument is declared, it is possible for the caller of the rule to override the default value by passing an explicit argument. You may not wish the caller to have any control over the value of the argument. This can be prevented by locking the argument. Arguments are locked by including the attribute `locked` with a value of `true` in the `RuleArgument` element (see Code Example 2-32):

Code Example 2-32 Locking an Argument

```
<Rule name='generateEmail' localScope='true'>
   <RuleArgument name='firstname'>
      <Comments>The first name of a user</Comments>
   </RuleArgument>
   <RuleArgument name='lastname'>
      <Comments>The last name of a user</Comments>
   </RuleArgument>
   <RuleArgument name='domain' value='example.com' locked='true'>
      <Comments>The corporate domain name</Comments>
   </RuleArgument>
   <concat>
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@</s>
      <ref>domain</ref>
   </concat>
</Rule>
```

In the preceding example, the argument `domain` is locked, which means its value will always be `example.com`, even if the caller tries to pass an value for the argument. If this rule is to be used at a site whose domain name is not `example.com`, all the administrator needs to do is edit the rule and change the value of the argument. There is no need to understand or modify the rule expression.

Securing Rules

You should secure a rule so it cannot be used in an unintended way if that rule contains sensitive information such as credentials or it calls out to a Java utility that can have dangerous side effects.

Securing rules is especially important if the rules are called from forms. Form rules run above the session, so exposed rules are available to anyone capable of creating a session — either through the API or a SOAP request.

This section provides the following information:

Securing a Rule

To secure a rule:

- **Place the rule in an appropriate organization**. Rules are often put in the organization named `All` so that everyone can access them. This is convenient for simple rules that perform calculations but have no side effects.
- **Do not put sensitive rules in All**. Instead, consider placing them in Top or another suitably high level organization so that only high level administrators can execute them directly.

Creating Rules that Reference More Secure Rules

When you call a rule, you are first authorized for that rule. If authorized, that rule can then call other rules without further checking authorization. This allows users to be given indirect access to secure rules. The user cannot view or modify the contents of the secure rule. They can call it only through a rule to which they have been given access.

To create a rule that references a more secure rule, the user creating the rule must control both of the organizations that contain the rules. Typically, the secure rule is in a high level organization such as `Top`. The insecure rules are then placed in a lower level organization to which more users have access.

Chapter 3

Working with Variable Namespaces

This chapter provides an overview of common Identity Manager tasks and processes, how they are typically used, and the namespace in which they run.

The information is organized as follows:

Active Sync

The following table provides information about the common Identity Manager processes or tasks related to the Active Sync category:

Table 3-1 Active Sync Processes/Tasks

Process or Task Running	How it is Used	Namespace
ActiveSync `IAPIUser`	• Processes user-related changes on a particular resource. • Performs actions directly on the full User view before launching the designated workflow process.	Merges attributes from the ActiveSync event into the User view. Typical attributes on the Input Form include: • `accounts[*].*` • `waveset.*` • `accountInfo.*` • `activeSync.`<*LHS Attr Name*> • `activeSync.`*resourceName* • `activeSync.resourceId` • `activeSync.resource` • `display.session` (session for Proxy Admin) • `global.`<*LHS Attr Name*> (if `set globals` flag is set on resource)
ActiveSync `IAPIProcess`	• Processes generic events on a resource by creating a Process view. • Top-level fields in Process view are arbitrary inputs to the task. • Collects attributes related to launching the task under the `global` attribute. • Writes the workflow to retrieve inputs from under `global` rather than as top-level attributes.	Launches the specified task with ActiveSync poll attributes dumped into top-level workflow `global` attribute. Workflow attributes assume the form: `global.`<*LHS Attr Name*>

Interactive Edits

The following table provides information about the common Identity Manager processes or tasks related to the interactive edits category:

Table 3-2 Interactive Edits Processes/Tasks

Process or Task Running	How it is Used	Namespace
Administrator Interface Forms	View/form interactions through the Administrator Interface JSPs for launching requests (no workflow has been launched yet) Does *not* apply to approval pages	The view is edited directly, so typical attribute names of the form: • `accounts[*].*` • `waveset.*` • `accountInfo.*` • `:display.session` (*session for admin*)
WorkItems	Launched using the `<ManualAction>` directive. Applies to both custom tasks and administrator approvals. The form associated with a specified workflow can set the base context to `variables.user`. This eliminates the need to put `user.variables` in the variable name.	The WorkItem is the name space, so typical attribute names of the form: • `complete (WorkItem attribute)` • `variables.* (task variables)` • `variables.<view>.accounts[*].*` • `variables.<view>.waveset.*` • `variables.<view>.accountInfo.*` • `:display.session` (*session for Owner*)
Role-defined Assigned Resource Attribute Value Rule	Rule is attached to Role definitions and evaluated when the view is refreshed to assign values to resource account attributes.	Regardless of the calling context, the rule is applied directly to the view. Consequently, expect typical view attribute names of the form: • `accounts[*].*` • `waveset.*` • `accountInfo.*`

Load Operations

The following table provides information about the common Identity Manager processes or tasks related to the load operations category:

Table 3-3 Load Operations Processes/Tasks

Process or Task	How it is Used	Namespace
Load from File	Retrieves account information from a CVS or XML file (invoked through Administrator Interface). Identity Manager reads a `WSUser` object from a file, converts it to the User view, and applies the form. The attributes are processed as if they were extended attributes of the Identity Manager user. Attributes are put in `accounts[Lighthouse]` and will only be put under the `global` attribute if the form defines global fields for each of them.	All attribute values for each line in the file are pulled into the global namespace: `global.<attr name>` **Note**: Applies to `create` operations only.
Load from Resource	Retrieves account information from a particular resource (invoked through Administrator Interface and uses an adapter to list and fetch accounts).	All attribute values for each account on the resource are pulled into the global namespace. `global.<LHS Attr Name>` **Note**: Applies to `create` operations only.
Bulk Operations	Retrieves commands and User view data from a CVS file (invoked through Administrator Interface). You can specify any attribute in the User view namespace. Attribute names are specified using the view path syntax. See “Understanding the User View” in the *Identity Manager Technical Deployment Overview* for more information about the User view namespace and view path syntax	Attribute values from the file are pulled into the global namespace: • `accounts[*].*` • `waveset.*` • `accountInfo.*` • `global.*` **Note**: There is no authorized session available.

NOTE Identity Attributes can now be applied during Load from File and Load from Resource operations when you add Load from File and Load from Resource to the list of enabled applications for the identity attributes.

When enabled for Load from File and Load from Resource, these pages do not display options for selecting a User Form, Update Attributes, or Merge Attributes. If you select Update Accounts, Identity Manager fully processes all identity attributes and re-provisions accounts. Otherwise, only those attributes that are sourced from the resource being loaded (and flow to the identity user) are processed.

During Reconciliation, Identity Manager applies identity attributes only for the following Reconciliation Responses:

- Create user based on resource account
- Create resource account for user

Reconciliation Rules

The following table provides information about the common Identity Manager processes or tasks related to the reconciliation rules category:

Table 3-4 Reconciliation Rules Processes/Tasks

Process or Task Running	How it is Used	Namespace
Correlation Rule	Invoked during reconciliation to associate a resource account with one (or more) Identity Manager users	All attribute values for the resource account defined in the schema are provided in the form `account.`*<LHS Attr Name>* Returns: • Matching Identity Manager user name • List of AttributeConditions or WSAttributes that are used to search for matching Identity Manager user
Confirmation Rule	Invoked during reconciliation if the Correlation Rule results in multiple matches. The resource account is compared against each correlated Identity Manager user.	All attribute values for the resource account and all attributes in the user view are provided in the form: • `account.`*<LHS Attr Name>* • `user.accounts[*].*` • `user.waveset.*` • `user.accountInfo.*` **Returns**: Logical true or false (`1` or `0`) depending on whether there is a match

NOTE Identity Attributes can now be applied during Reconciliation operations if you add Reconciliation to the list of enabled applications for the identity attributes.

When enabled for Reconciliation, these pages do not display options for selecting a User Form, Update Attributes, or Merge Attributes. If you select Update Accounts, Identity Manager fully processes all identity attributes and re-provisions accounts. Otherwise, only those attributes that are sourced from the resource being loaded (and flow to the identity user) are processed.

During Reconciliation, Identity Manager applies identity attributes only for the following Reconciliation Responses:

- Create user based on resource account
- Create resource account for user

SPML

The following table provides information about the common Identity Manager processes or tasks related to the SPML category:

Table 3-5 SPML Processes/Tasks

Process or Task Running	How it is Used	Namespace
Person object class	Generic implementation of SPML interface. SPMLPerson Form, identified in SPML Configuration object, specifies mapping from a flat namespace of SPML schema to view attributes.	Pairs of mapping fields provided in form. Fields with • `<Derivation>` expressions set response schema attribute to view attribute. Fields with Derivations will have flat names, but reference view paths in their derivation expression. • `<Expansion>` expressions push request schema attribute to view attribute. Fields with Expansions will have path names, but reference flat names in their Expansion expression. The namespace for the view attribute consists of the `accounts`, `waveset`, `accountInfo` namespace attributes. The namespace of SPML schema attributes consists of a flat namespace.
Any request with form parameter set to **view**	No form processing	View attributes are set directly: • `accounts[*].*` • `waveset.*` • `accountInfo.*`

X.509 Integration

The following table provides information about the common Identity Manager processes or tasks related to the X.509 integration category:

Table 3-6 X.509 Integration Processes/Tasks

Process or Task Running	How it is Used	Namespace
Login Correlation Rule	Provides mechanism for resolving conflicting Identity Manager user entries (rule incorporates standard X.509 certificate).	Provide standard certification fields plus critical and non-critical extension properties. Certification properties assume the form `cert.<field name>.<subfield name>`: • `cert.subjectDN` • `cert.issuerDN` **Note**: There is no authorized session available. **Returns**: • AttributeCondition • list of AttributeConditions
New User Name Rule	If no user is correlated using Login Correlation Rule, provides mechanism for setting the name for a new Identity Manager user from the certification information.	See Login Correlation Rule **Returns**: `name` (or `accountId`) to use for the Identity Manager user

Miscellaneous Variable Contexts

The following table provides information about the common Identity Manager tasks or processes related to the miscellaneous variable contexts category:

Table 3-7 Miscellaneous Variable Contexts Processes/Tasks

Process or Task Running	How It is Used	Namespace
Launch Forms	Embedded in a TaskDefinition for the purpose of initializing the Executor	Any field elements specified are assimilated directly into the task context, and, if launching a workflow task, are available as top-level variables.
User Members Rule	Defined specifically for organizations that must dynamically return the list of member users. This rule cannot fetch against the repository. Instead, it is limited to `FormUtil.getResourceObjects` calls, such as finding all the entries in a specified directory OU.	User view of authenticated administrator (*no resource account attributes fetched*) plus the administrator's session: • `accounts[Lighthouse].*` • `waveset.*` • `accountInfo.*` • `context` (*authenticated administrator's session*)

Chapter 4

Developing Adapters

This chapter describes how to construct your own custom Sun Java™ System Identity Manager resource adapter that is tailored to your company or customer. Specifically, this chapter:

- Identifies mechanisms you can use to establish and maintain connections between Identity Manager and external resources
- Provides an introduction to resource adapter Java file structure
- Describes how to create, test, and load a resource adapter

This information is organized as follows:

You are encouraged to use the sample adapters (called *skeleton adapters*) that ship with the product as a starting point for your own custom adapter. To enhance your understanding of these valuable starter files, this chapter uses them frequently to exemplify particular characteristics of resource adapter files.

Overview

This section provides an overview of basic issues related to adapter purpose and function with Identity Manager. It covers the following topics:

Who Should Read this Chapter?

This chapter provides background that describes overall resource adapter design and operation that may be useful to:

- Developers working to create their own custom resource adapter
- Identity Manager administrators who are learning how the Identity Manager system works or how to troubleshoot problems with resource adapters.

This chapter assumes that you are familiar with creating and using the built-in Identity Manager resources and familiar with the topics discussed in the Resources chapter of *Sun Java™ System Identity Manager Administration*.

What is a Resource Adapter?

Each resource that you want Identity Manager to communicate with or manage must be defined by a *resource object* in Identity Manager. Identity Manager requires a resource object for each resource that it must manage. A resource adapter class contains the methods needed to register the resource object in the Identity Manager repository and to manage the external resource.

An adapter's primary purpose is to define the essential characteristics of this resource type. This information is saved in the Identity Manager repository as a resource object.

Resource objects define the capabilities and configuration of the resource you are managing in Identity Manager, including the information described in Table 4-1.

Table 4-1 Information Defined by Resource Objects

Type of Information	Sample Attributes
Connection information	• Host name • Administrative account name • Administrative account Password
User attributes	• First name • Last name • Phone numbers
Identity Manager attributes	• List of approvers • Password policy for the resource • How many times to repeat attempts when contacting the resource

Your customized adapter provides the means to define these essential characteristics as well as the methods to transform requests from Identity Manager into actions performed on the resource.

NOTE You can view resource objects from the Identity Manager Administrator Interface from the Debug page.

A resource adapter serves as a proxy between Identity Manager and an external resource, such as an application or database. A resource adapter class implements the methods needed to push information from Identity Manager to the resource or optionally pull information from the resource into Identity Manager. The optional pull capability is known as Active Sync. A resource adapter with Active Sync capability is referred to as Active Sync-enabled.

Identity Manager supplies Active Sync-enabled adapters for some common resources, such as Active Directory, and supplies standard resource adapters for many other resource types, including Web servers, Web applications, databases, and even legacy applications and operating systems. The resource adapters supplied with Identity Manager provide a generic interface to each of the different resource types that Identity Manager supports. Given an instance of one of these supported resource types, the resource adapter can communicate with and manage the real-world resource.

The open architecture of Identity Manager allows for custom resource adapters in order to permit management of external resources that are not already supported by supplied resource adapters. The mechanism for creating new custom resource adapter types are discussed later in this chapter.

How Do Active Sync-Enabled Adapters and Standard Adapters Differ?

A resource adapter can be a standard adapter and/or an Active Sync-enabled adapter. In Java terms, a resource adapter extends the `ResourceAdapterBase` class to be a standard resource adapter, and implements the Active Sync interface to be Active Sync-enabled.

Active Sync-enabled adapters can retrieve information from the source. They are typically either driven by events or poll the resource for information. Standard resource adapters push changes to account information to the external resource managed by Identity Manager.

Standard resource adapters propagate changes from Identity Manager to their managed resources. The typical administrative activities that resource adapters perform include:

- Create, delete, or modify a user
- Enable, disable, and get a user
- Manage objects such as group membership or directory organization structure
- Authenticate users
- Connect to and disconnect from resource

Active Sync-enabled adapters can gather the data changes directly from the resource to initiate activities in Identity Manager. These activities include:

- Polling or receiving event notification of a change
- Issuing an action to create, update, or delete the resource account
- Editing or creating a user with a custom form
- Saving the changes to the resources
- Logging information about the progress and any errors that occur

Types of Resources Monitored by Active Sync-Enabled Adapters

Active Sync-enabled adapters are especially suitable for supporting the following resource types:

- **Applications with audit or notification interfaces**. Microsoft Active Directory and PeopleSoft, for example, have external interfaces that you can configure to notify another application or add an event to an audit log when specified changes occur.

  For example, when a user account is modified natively on the Active Directory server, the transaction is recorded in an audit log. The Identity Manager Active Directory resource can be configured to review this log every 30 minutes and any changes found will trigger events in Identity Manager.

  Other Active Sync-enabled adapters can be registered with the resource through an API. The adapter is then notified with an event message when changes occur to items. These messages contain a reference to the item that has changed, the information that is updated, and often the user that made the change. For example, the LDAP resource uses the notification implementation of Active Sync-enabled adapters.

- **Databases populated with update information**. Database resources can also be managed by using a table of deltas. You can generate this table through several different ways. For example, you could compare a snapshot of the database to the current values and creating a new table with the differences. The adapter can then pull the rows from the table of deltas, process them, and subsequently mark them when completed.

- **Databases with modification timestamps**. A third type of Active Sync-enabled adapter is one that can query for database entries that have been modified after a particular time, run updates, and then poll for new queries. By storing the last successfully processed row, Identity Manager can perform a “starts with” query to minimize polling impact. Only the changes that have been made to the resource because the last time are returned for processing.

How Standard Resource Adapters Work

This section presents an overview of the basic steps of standard adapter processing.

All standard resource adapters follow these general steps when pushing information from Identity Manager to the resource managed by Identity Manager:

1. Identity Manager server initializes the resource manager.

   All available resource types are registered through the Resource Adapter interface. As part of the registration process, the resource adapter provides a prototype XML definition.

2. User initiates process of creating a new resource.

   When an Identity Manager administrator creates a new resource, the task which is creating the form to display the prototype definition for the resource type is queried for the resource attribute fields. Identity Manager uses these attributes to display a form on the browser. The user who is creating the new resource fills in the information and clicks Save.

3. Identity Manager saves the entered information along with the other resource fields in the resource object repository under the name of the new resource object.

   When the user clicks Save during resource creation, the creation task gathers the entered data, executes any necessary validation, then serializes the data via XML before writing the serialized object to the object repository.

4. Identity Manager displays the list of available resources in a multi-selection box when an Identity Manager user is created, or modified.

   Selecting a resource causes Identity Manager to query the resource object for the available account attribute fields. Identity Manager uses these field descriptions to display a form that contains the attribute fields, which the user can fill in with the appropriate data.

5. The resource object is queried for the connection information when this form is saved, and a connection is established with the resource.

6. The adapter sends the command to perform the intended action on the account on the resource over this connection.

7. If this is a create request, the Identity Manager user object is updated with the resource account information.

When user account information is displayed, Identity Manager requests the list of resources on which the user has accounts from the saved account object. For each resource, Identity Manager queries the resource object and uses the connection information to establish a connection to the resource.

The adapter sends a command over this connection to retrieve account information for the user, and it uses the retrieved information to fill in the attribute fields that are defined in the resource object. The system creates a form to display these values.

How Active Sync-Enabled Adapters Work

This section describes:

- Basic Active Sync-Enabled Adapter Processing
- Identifying the Identity Manager User

Basic Active Sync-Enabled Adapter Processing

All Active Sync-enabled adapters follow the following basic steps when listening or polling for changes to the resource managed by Identity Manager. When the adapter detects that a resource has changed, the Active Sync-enabled adapter:

1. Extracts the changed information from the resource.
2. Determines which Identity Manager object is affected.
3. Builds a map of user attributes to pass to the `IAPIFactory.getIAPI` method, along with a reference to the adapter and a map of any additional options, which creates an Identity Application Programming Interface (`IAPI`) object.
4. Sets the logger on the IAPI event to the adapter's Active Sync logger.
5. Submits the `IAPI` object to the Active Sync Manager.
6. Active Sync Manager processes the object and returns to the adapter a `WavesetResult` object that informs the Active Sync-enabled adapter if the operation succeeds. This object can contain many results from the various steps that the Identity Manager system uses to update the identity. Typically, a workflow also handles errors within Identity Manager, often ending up as an Approval for a managing administrator.

Exceptions are logged in the Active Sync and Identity Manager tracing logs with the `ActiveSyncUtil.logResourceException` method.

Identifying the Identity Manager User

When the Active Sync-enabled adapter detects a change to an account on a resource, it either maps the incoming attributes to an Identity Manager user, or creates an Identity Manager user account if none can be matched.

The following parameters determine what happens when a change is detected.

Table 4-2 Active Sync-Enabled Adapter Parameters

Parameter	Description
Process Rule	Either the name of a TaskDefinition, or a rule that returns the name of a TaskDefinition, to run for every record in the feed. The process rule gets the resource account attributes in the Active Sync namespace, as well as the resource ID and name. A process rule controls all functionality that occurs when the system detects any change on the resource. It is used when full control of the account processing is required. As a result, a process rule overrides all other rules. If a process rule is specified, the process will be run for every row regardless of any other settings on this adapter. At minimum, a process rule must perform the following functions: • Query for a matching User view. • If the User exists, checkout the view. If not, create the User. • Update or populate the view. • Checkin the User view. It is possible to synchronize objects other than User, such as LDAP Roles.
Confirmation Rule	Rule that is evaluated for all users returned by a correlation rule. For each user, the full user view of the correlation Identity Manager identity and the resource account information (placed under the “account.” namespace) are passed to the confirmation rule. The confirmation rule is then expected to return a value which may be expressed like a Boolean value. For example, “true” or “1” or “yes” and “false” or “0” or null. For the Database Table, Flat File, and PeopleSoft Component Active Sync adapters, the default confirmation rule is inherited from the reconciliation policy on the resource. The same confirmation rule can be used for reconciliation and Active Sync.

Table 4-2 Active Sync-Enabled Adapter Parameters *(Continued)*

Parameter	Description
Correlation Rule	If no Identity Manager user's resource information is determined to own the resource account, the Correlation Rule is invoked to determine a list of potentially matching users/accountIDs or attribute conditions, used to match the user, based on the resource account attributes (in the account namespace). Returns one of the following types of information that can be used to correlate the entry with an existing Identity Manager account: • Identity Manager user name • `WSAttributes` object (used for attribute-based search) • List of `AttributeCondition` or `WSAttribute`-type items (`AND`-ed attribute-based search) • List of `String`-type items (each item is the Identity Manager ID or the user name of an Identity Manager account) If more than one Identity Manager account can be identified by the correlation rule, a confirmation rule or resolve process rule is required to handle the matches. For the Database Table, Flat File, and PeopleSoft Component Active Sync adapters, the default correlation rule is inherited from the reconciliation policy on the resource. The same correlation rule can be used for reconciliation and Active Sync.
Delete Rule	A rule that can expect a map of all values with keys of the form `activeSync.` or `account.` pulled from an entry or line in the flat file. A LighthouseContext object (`display.session`) based on the proxy administrator's session is made available to the context of the rule. The rule is then expected to return a value which may be expressed like a Boolean value. For example, "true" or "1" or "yes" and "false" or "0" or null. If the rule returns true for an entry, the account deletion request will be processed through forms and workflow, depending on how the adapter is configured.
Resolve Process Rule	Name of the TaskDefinition or a rule that returns the name of a TaskDefinition to run in case of multiple matches to a record in the feed. The Resolve Process rule gets the resource account attributes as well as the resource ID and name. This rule is also needed if there were no matches and Create Unmatched Accounts was not selected. This workflow can be a process that prompts an administrator for manual action.
Create Unmatched Accounts	If set to `true`, creates an account on the resource when no matching Identity Manager user is found. If `false`, the account is not created unless the process rule is set and the workflow it identifies determines that a new account is warranted. The default is `true`.
Populate Global	If set to `true`, populates the global namespace in addition to the ActiveSync namespace. The default value is `false`.

The presence of a Process Rule determines whether `IAPIProcess` is used, or `IAPIUser` is attempted. If Identity Manager cannot use `IAPIUser` because a correlation or confirmation rule did not uniquely identify an Identity Manager user for the event given the other parameter settings, and a Resolve Process Rule is configured, it is used to create an `IAPIProcess` event. Otherwise an error condition is reported.

`IAPIUser` checks out a view and makes this view available to the user form. For creates and updates, the `User` view is checked out. For deletes, the `deprovision` view is checked out. However, a view is not available with `IAPIProcess`. Either a Process rule has been set, or a Resolve Process rule is invoked.

Preparing to Create a Custom Adapter

The following section describes an approach to begin developing your custom adapter.

- Understanding Your Resource
- What's in the Resource Extension Facility (REF) Kit?
- Getting Started

Understanding Your Resource

This section helps you prepare to edit the adapter file, including:

- Getting to Know Your Resource
- Additional Preparations

Getting to Know Your Resource

Use the following questions to help gather information and define prerequisites.

Which type of default Identity Manager resource adapter does your planned adapter most closely resemble?

See Table 4-4 on page 179 for a brief description of the adapter files that are shipped with the standard Identity Manager configuration. Select the Identity Manager adapter file that most closely matches the resource type to which you are connecting.

What are the characteristics of user accounts on the resource? How do you perform basic tasks on the resource?

You should find out how to perform the following tasks on the remote resource:

- Authenticate access to the remote resource
- Update user
- Search for changes to users (Active Sync-enabled only)
- Get the details on the changed users
- List all users on the system
- Identify a way to search for changed users only. (Active Sync-enabled only)
- List other system objects such as groups, which are used in the `listAllObjects` method

You will also need to identify the minimum attributes needed to perform an action as well as all supported attributes.

Do you have appropriate tools to support connection to the resource?

Many resources are shipped with a set of APIs or a complete toolkit that you can use when integrating an outside application to the resource. Find out if your resource has a published set of APIs or if the toolkit provides documentation and tools to speed up integration with Identity Manager. For example, for databases, you will need to connect to the database through JDBC.

Who can login and search for users on the resource? How do they do so?

Most resource adapters require an administrative account that they can run as to perform tasks such as searching for users and retrieving their attributes. This is often a highly privileged, or super user, account but may be a delegated administration account with read-only access.

Can the resource be customized to add additional attributes?

Most resources allow you to extend the built-in attributes. For example, Active Directory and LDAP both allow the creation of custom attributes.

Consider which attributes you want this resource to maintain in Identity Manager, what their names are on the resource and how you want them named in Identity Manager. The attribute names will go in the schema map and end up as inputs to the form that is used when creating a resource of that type.

Which attributes/actions on the source will create an event? (Active Sync-Enabled)

If the resource supports subscribing to notification messages when changes are made (such as LDAP), identify which attribute changes you want to trigger the notification and which attributes you want in the message.

Which action will Identity Manager take when an event on the source is detected? (Active Sync-enabled)

These actions can include:

- Create, update, or delete a user
- Disable or enable accounts
- Update the answers used to authenticate a user
- Update a phone number

Do you want your Active Sync-enabled adapter to be driven by events in the external resource or by set polling intervals?

Before you make your decision, consider how polling works within typical Identity Manager installations. Although some installations implement or are driven by external events, most environments in which Identity Manager is deployed use a hybrid method.

You can choose one of the following approaches:

- **Setting Up Polling Intervals**. The poll interface can be called by an `ActiveSyncManager` thread at a configurable interval or on a specified schedule. You can set polling parameters, including settings such as faster polling if work was received, thread-per-adapter or common thread, and limits on the amount of concurrent operations.

- **Setting Up an Event-Driven Environment**. Adapters can also be purely event-driven. In this model, the adapter sets up a listening connection (such as an LDAP listener) and waits for messages from the remote system. In this case, the `poll` method can be implemented to do nothing, and the polling interval set to an arbitrary value, such as once a week. If the updates are event-driven, they must have a guaranteed delivery mechanism (such as MQ Series) or synchronization will be lost.
- **Implementing a Hybrid Solution**. In this scenario, external events can trigger smart polling, and the regular poll routine can recover from missed messages. (*Smart polling* is adapting the poll rate to the change rate and polling infrequently unless changes are being made often. It balances the performance impact of frequent polling with the update delays of infrequent polling.)

  In this model, incoming messages can be queued and multiple updates for a single object can be collapsed into a single update, increasing efficiency. For example, multiple attributes can be updated on a directory, and each one triggers a message. The poll routine can walk the message queue and remove all duplicates, then fetch the complete object. This ensures that the latest data is synchronized and the updates are handled efficiently.

Additional Preparations

After using the preceding questions to profile your resource, continue your preparations by completing the following steps:

1. Identify the list of methods that you must write.

   The most time-intensive part of creating an adapter file is writing your own methods for pushing information from Identity Manager to the resource or creating a feed from the resource into Identity Manager.

   The methods you include in your Java file depend in part upon which type of resource you are creating a connection to.

2. Review the Identity Manager JavaDoc.

   The relevant JavaDoc are included on your product CD and identify the base classes and methods that you will use as is or those that you will need to extend.

What's in the Resource Extension Facility (REF) Kit?

The Sun Resource Extension Facility Kit (provided in the /REF directory on your product CD or with your install image) is your guide to creating your custom resources. This REF Kit provides the example code and tools you need to create custom resource and Active Sync-enabled adapters.

Table 4-3 describes the file and directory categories provided in the REF Kit.

Table 4-3 REF Kit Files and Directories

Component	Location	Description
waveset.properties	REF/config	Contains a copy of the properties file you need when testing your custom adapter.
Javadocs	REF/javadoc	Contains the generated JavaDoc that describe the classes you should work with when writing a custom adapter. To begin viewing these JavaDoc, point your browser to: REF/javadoc/index.html
source files	REF	Provides several sample resource adapter source files to use as a guide when creating your own adapter.
Test source files	REF/test	Contains sample resource adapter test source files to use as a basis for your custom adapter.
lib	REF/lib	Contains the jar files you need to compile and test a custom adapter.
Before You Begin.README	REF	Provides a one-page outline describing how to customize an adapter.
Design-for-Resource-Adapters.htm		Introduces the basic architecture of a resource adapter.

Resource Adapter Samples

The Ref kit provides sample adapter files and tools that you can use to jump-start the process of creating your own custom adapter. The resource source files give you an example of the different types of resources that are commonly developed. The Ref kit also provides a set of skeleton files that are intended for you to use as the basis for your adapter development.

Table 4-4 describes the sample files provided in the REF Kit.

Table 4-4 Resource Adapter Sample Files

Type of Sample File	File Name
Database accounts	`MySQLResourceAdapter.java`
Database tables	`ExampleTableResourceAdapter.java*`
File-based accounts	`XMLResourceAdapter.java`
LDAP accounts	Two examples for developing custom LDAP resource adapters: • For simple methods, base the adapter on `LDAPResourceAdapter.java`. • For complex changes, base the adapter on `LDAPResourceAdapterBase.java`.
UNIX accounts	`AIXResourceAdapter.java`
Skeleton files	• For standard resources: `SkeletonStandardResourceAdapter.java` • For standard and Active Sync-enabled resources: `SkeletonStandardAndActiveSyncResourceAdapter.java` • For Active Sync-only resources: `SkeletonActiveSyncResourceAdapter.java` To create unit tests for a custom adapter, use `test.SkeletonResourceTest.java` file

NOTE For table-based resources, use the Resource Adapter Wizard to create the adapter instead of writing a custom adapter.

See the "Configuration" chapter in *Identity Manager Administration* for more information about using this wizard to customize adapters shipped by Sun.

Identifying Code for Revision

Both skeleton files that are shipped in the REF kit contain `@todo` and `change-value-here` text strings that identify the places in the code where you must make adapter-specific modifications. When customizing these files, search on these strings to determine where you must make changes.

Table 4-5 identifies the text you can search for when identifying code that needs customization.

Table 4-5 Search Text Strings

Comment	Description
`@todo`	Identifies a method that you must rewrite to handle the particular scenario you are supporting.
`change-value-here`	Identifies a substitution that you must make.

Setting Up a Build Environment

Set up a build environment following these steps:

Prerequisite: You must install the JDK version required for your Identity Manager version.

After installing the JDK, install the REF kit by copying the entire REF directory to your system.

1. Change directories to the new directory and create a file called `ws.bat` (if you are working on a system running a Microsoft Windows operating system) or `ws.sh` (if you are working on a UNIX system).
2. Add the following lines to this file before saving and closing it:

:

```
If you are using a ws.bat file, include these lines:
set WSHOME=<Path where REF is installed>
set JAVA_HOME=<Path where REF is installed>
set PATH=%PATH%;%JAVA_HOME%\bin

If you are using a ws.sh file, include these lines:
WSHOME=<path where REF is installed>
JAVA_HOME=<path where jdk is installed>
PATH=$JAVA_HOME/bin:$PATH

export WSHOME JAVA_HOME PATH
```

Where `WSHOME` is the path where the REF kit is installed and `JAVA_HOME` identifies the path where Java is installed.

Getting Started

Use the following steps to start creating a custom adapter:

1. Copy the appropriate skeleton file to a name of your choice.
   - For standard resources: `SkeletonStandardResourceAdapter.java`
   - For standard and Active Sync-enabled resources: `SkeletonStandardAndActiveSyncResourceAdapter.java`
   - For Active Sync-only resources: `SkeletonActiveSyncResourceAdapter.java`
2. Edit your adapter source file. Perform a global search for `Skeleton`, and replace it with the name of your adapter.
3. Source the `ws.bat` or `ws.sh` file you created previously to set up your environment.
4. Compile your source files using the following command:

   ```
   javac -d . -classpath %CLASSPATH% yourfile.java
   ```

Getting Familiar with Your Resource Adapter

This section provides an overview of the resource adapter source code components that are common to most adapters and provides examples from the skeleton adapters that are part of the Resource Extension Facility.

Adapter Components

Table 4-6 describes the main resource adapter components.

Table 4-6 Adapter Components

File Component	Description
Standard Java header information	Includes identification of the parent class for the new adapter class file you are creating, constructors, and imported files.
prototypeXML string	Defines which information about the resource appears in the Identity Manager interface. Maps each resource attribute to an Identity Manager User Attribute.
Methods that create a feed from the resource into Identity Manager	In Active Sync-enabled adapters, presents methods that search the resource for changes or receive updates. Writing these methods requires that you understand how to register for or search for changes on the resource as well as how to communicate with the resource.
Methods that perform the update operations in the Identity Manager repository	In Active Sync-enabled adapters, performs the feed form the resource into Identity Manager.
Methods that write information from Identity Manager in to the external resource	These customized methods must be written by a developer with knowledge of the resource.

NOTE One enhancement to the skeleton adapter file is the addition of the `@todo` and `@change-attribute-here` comments. These comments mark places where you should substitute adapter-specific text.

Resource Information Defined in Resource Adapters

There are three distinct types of information that define a resource in Identity Manager, and they are defined for the resource type in the adapter file:

- **Resource attributes**. Defined in the `prototypeXML` section of the adapter Java file. You can modify these values from the Identity Manager interface when creating a specific instance of this resource type.
- **Account DN or identity template**. Includes account name syntax for users, which is especially important for hierarchical namespaces.
- **Identity Manager account attributes**. Defined in the Schema map for the resource. Identity Manager account attributes define the Identity Manager attributes of the resource object.

Standard Java Header Information

The header information is representative of the standard Java files (including public class declaration and class constructors). You must edit the sections of the file that list the constructors and public classes, and, if necessary, the imported files.

PrototypeXML Section

This section of the adapter Java file (the `prototypeXML`) is the XML definition of a resource. It must contain the resource name and all attributes of the resource that the Identity Manager interface will display.

Table 4-7 introduces the components of the prototypeXML. Some components are more relevant for resource adapters than Active Sync-enabled adapters. Each component that is relevant for Active Sync-enabled adapters is discussed in greater detail in the sections following the table.

Table 4-7 prototypeXML Components

Component	Description
Resource	Define top-level characteristics of resource. Keywords for this element include: • **`syncSource`**: If true, then adapter must be Active Sync-enabled. • **`facets`**: Specifies the modes enabled for this resource. Valid values are • **`provision`**: Provisioning is enabled. Both modes are available if syncSource is true. • **`activesync`**: Active Sync only

Table 4-7 prototypeXML Components *(Continued)*

Component	Description
Resource attributes	Used by Identity Manager to configure the resource. Resource attributes are XML elements that are defined with the `<ResourceAttribute>` element. Keywords for this element include: • **type**: Identifies supported data types (currently, string only). • **multi**: Specifies whether multiple values can be accepted for the attribute. If set to `true`, a multi-line box is displayed. • **facets**: Specify usage for this resource attribute. • **provision**: Used in standard processing. This is the default value. • **activesync**: Used in Active Sync processing, if Active Sync is configured and enabled.
Account attributes	Defines the default schema map for basic user attributes. Account attributes are defined with the `<AccountAttribute>` element. You define account attributes that map to the standard Identity Manager account attribute types differently than you map to custom attributes. For more information on mapping account attributes to resource attributes, see "How to Map Resource Attributes to Identity Manager Account Attributes."
Identity template	Defined by the `<Template>` tag, this template defines how the account name for the user is built. Account names are typically of two forms. One form is the `accountId`, which is typically used for resources with a flat namespace such as NT or Oracle. The other is a complete distinguished name of the user in the form: `cn=accountId,ou=sub-org,ou=org,o=company`. This latter form is used for hierarchical namespaces such as directories.
Login configuration	(Standard resource adapter only) Defined by the `<LoginConfigEntry>` element, this value defines the values for support of pass-through authentication for this resource. For more information about pass-through authentication, see the *Sun Java™ System Identity Manager Resources Reference.*
Form	(Active Sync-enabled adapter only) Designates a form object that will process the data from the Active Sync-enabled adapter before it is integrated into Identity Manager. A form is optional, but in most cases it provides flexible changes in the future. It can be used to transform the incoming data, map the data to other user attributes on the other resource accounts, and cause other actions in Identity Manager to occur.
Resource user form	(Active Sync-enabled adapter only) Defined by the `<ResourceUserForm>` element.

Resource Methods

Resource methods can be categorized by task. When developing your own custom adapter, you first determine which categories are necessary to meet the goals of your development. For example, is your adapter going to be an Active Sync-enabled adapter? Will the first phase of your deployment support password reset only? The answers to these questions determine which methods must be completed. Additional information about each functional category is discussed later in this chapter.

These resource methods categories are described in Table 4-8.

Table 4-8 Resource Methods Categories

Category	Description
Basic	Provide the basic methods for connecting to the resource and performing simple actions
Bulk operations	Provide bulk operations to get all the users from the resource
Active Sync	Provides the methods to schedule the adapter.
Object management	Provides the methods to manage groups and organizations on your resources.

Defining the PrototypeXML Section

In the resource source code, the prototypeXML defines the resource object that is stored in the Identity Manager repository.

Table 4-9 describes the five, distinct types of information that define a resource in Identity Manager. The first three types are visible to administrators when they are defining the resource. The remaining two types help define the resource but are not easily modifiable by Identity Manager administrators.

Table 4-9 Information Types that Define Identity Manager Resources

Information Type	Description
Resource attribute	Defines the connection information on the resource being managed. Resource attributes typically include the resource host name, resource administrator name and password, and the container information for directory-based resources. Identity Manager attributes such as the list of resource approvers and the number of times to retry operations on the resource are also considered resource attributes.
Account attribute	Maps the relationship between the Identity Manager attribute name and the resource attribute name. These attributes are defined on the resource schema map and appear in the prototypeXML. For example, on LDAP, the resource attribute `sn` is mapped to the more common attribute in Identity Manager called `lastname`.
Identity template or account DN	Defines the default account name for users.
Login config	Defines the parameters that are used if this resource is to be used for pass-through authentication. Typically, these parameters are `username` and `password`. However, in the example of SecurId, it includes a user name and passcode.
Object management	

Resource Attributes

Resource attributes define:

- Resources you want to manage, along with other connection and resource characteristics.

  From the perspective of an administrator using the Identity Manager Administrator Interface, these attributes define the field names that are visible in the Identity Manager interface and prompt the user for values.

  For Windows NT resources, attributes can include source name, host name, port number, user, password, and domain. For example, the Create/Edit Resource page for a resource type requires a host field in which administrators creating a resource identify the host on which the resource resides. This field (not the contents of the field) is defined in this adapter file.

- Authorized account that has rights to create users on the resource. For a Windows NT resource, this includes the user and password fields.

- Source attributes including the form, the Identity Manager administrator that the adapter will run as, scheduling and logging information, and additional attributes used only in Active Sync methods.

How Resource Attributes Are Defined Resource attributes are defined in the Java file with the <ResourceAttribute> element as follows:

```
<ResourceAttribute name='"+RA_HOST+"' type='string' multi='false'\n"+
description='&lt;b&gt;host&lt;/b&gt;&lt;br&gt;Enter the resource host
name.'>\n"+
```

The description field identifies the item-level help for the RA_HOST field and must not contain the < character. In the preceding example, the < characters are replaced by < and '.

Table 4-10 describes the keywords you can use in <ResourceAttribute> element.

Table 4-10 <ResourceAttribute> Element Keywords

Keyword	Description
name	Identifies the name of the attribute. For a list of the required attributes, see "Required Resource Attributes."
type	Identifies data type used. Currently, only the string type is supported.
multi	Specifies if multiple values can be accepted for the attribute. If true, a multi-line box is displayed.

Table 4-10 <ResourceAttribute> Element Keywords *(Continued)*

Keyword	Description
description	Identifies the item-level help for the RA_HOST field. Identity Manager displays help with the item being described (host in this case) in bold text. Because the HTML brackets necessary to do this (< and >) interfere with XML parsing, they are replaced by < and >. After the binary is translated, the description value looks like: Description='host Enter the resource host name.'
facets	Specifies the usage of this resource attribute. Valid values are • **provision**: Used in standard processing. This is the default value. • **activesync**: Used in Active Sync processing for an Active Sync-enabled adapter.

Overwriting Resource Attributes You can choose between the following two strategies when working with resource adapters and adapter parameters:

- Use the adapter's Attribute page to set a resource attribute value once for all users
- Set a default attribute value on the adapter, then subsequently override its value, as needed, within your user form

In Code Example 4-1, the user form must override the resource attribute value for template during the creation of each user. When implementing similar code in a production environment, you would probably include more detailed logic that would calculate this template value within your user form.

Code Example 4-1 Overriding the Resource Attribute Value for template

```
<Field name='template'>
   <Display class='Text'>
      <Property name='title' value='NDS User Template'/>
   </Display
</Field>

<!-- Change NDS for the name of your NDS resource -->
<!-- The word Template is the name of the attribute field, as viewed in the
resource xml -->
<Field name='accounts[NDS].resourceAttributes.Template'>
   <Expansion>
      <ref>template</ref>
   </Expansion>
</Field>
```

Required Resource Attributes Table 4-11 describes the resource attributes provided in the skeleton adapter files.

Table 4-11 Resource Attributes in Skeleton Adapter Files

Required Resource Attribute	Description
RA_HOST	The host name of the resource. This corresponds to the Host field on the Resource Parameters page.
RA_PORT	The port number used to communicate with the resource. This corresponds to the Port field on the Resource Parameters page.
RA_USER	The name of a user account that has permission to connect to the resource. The field name varies on the Resource Parameters page.
RA_PASSWORD	The password for the account specified by RA_USER. This corresponds to the Host field on the Resource Parameters page.

Table 4-12 describes the Active Sync-specific attributes defined in the ACTIVE_SYNC_STD_RES_ATTRS_XML string of the Active Sync class.

Table 4-12 Active Sync-Specific Attributes Defined in ACTIVE_SYNC_STD_RES_ATTRS_XML

Required Resource Attribute	Description
RA_PROXY_ADMINISTRATOR	Identity Manager administrator for authorization and logging. This corresponds to the Proxy Administrator field in the Identity Manager display. You do not define this value in the adapter Java file. Instead, an administrator enters this information when defining a specific instance of this resource type.
RA_FORM	Form that processes incoming attributes and maps them to view attributes. This corresponds to the Input Form field.
RA_MAX_ARCHIVES	Specifies the number of log files to retain. • If you specify **0**, then a single log file is re-used. • If you specify **-1**, then log files are never discarded.
RA_MAX_AGE_LENGTH	Specifies the maximum time before a log file is archived. If time is zero, then no time-based archival will occur. If the value of RA_MAX_ARCHIVES is zero, then the active log will instead be truncated and re-used after this time period.
RA_MAX_AGE_UNIT	One of seconds, minutes, hours, days, weeks, or months. This value is used with RA_MAX_AGE_LENGTH.
RA_LOG_LEVEL	Logging level (0 disabled; 4 very verbose). This corresponds to the Log Level field in the Identity Manager display.
RA_LOG_PATH	Absolute or relative path for the log file. This corresponds to the Log File Path field in the Identity Manager display.

Table 4-12 Active Sync-Specific Attributes Defined in ACTIVE_SYNC_STD_RES_ATTRS_XML *(Continued)*

Required Resource Attribute	Description
RA_LOG_SIZE	Maximum log file size. This corresponds to the Maximum Log File Size field in the Identity Manager display.
RA_SCHEDULE_INTERVAL	Pop-up menu of the supported scheduling intervals (second, minute, hour, day, week, month).
RA_SCHEDULE_INTERVAL_COUNT	The number of intervals between scheduled periods (for example, 10 minutes would have an interval count of 10 and an interval of minute). Not necessary for Active Sync-enabled adapters.
RA_SCHEDULE_START_TIME	The time of the day to run. Setting this to 13:00 and the interval to week would set the adapter to run at 1 P.M. once a week. Not necessary for Active Sync-enabled adapters.
RA_SCHEDULE_START_DATE	The date to start scheduling. Setting date to `20020601`, the interval to `month`, and the time to `13:00` would have the adapter start running on June 1st and run once a month at 1 P.M. Not necessary for Active Sync-enabled adapters.

Table 4-13 describes the Active Sync-specific attributes which are defined in the ACTIVE_SYNC_EVENT_RES_ATTRS_XML string in the by default class. See "Identifying the Identity Manager User" for more information about each of these attributes.

Table 4-13 Active Sync-Specific Attributes Defined in ACTIVE_SYNC_EVENT_RES_ATTRS_XML

Required Resource Attribute	Description
RA_PROCESS_RULE	The name of a TaskDefinition or a rule that returns the name of a TaskDefintion to run for every record in the feed. This parameter overrides all others.
RA_CORRELATION_RULE	A rule that returns a list of strings of potentially matching users/accountIDs, based on the resource account attributes in the account namespace.
RA_CONFIRMATION_RULE	A rule that confirms if a user is a match.
RA_DELETE_RULE	A rule that determines whether a delete detected on the resource is processed as an IAPI delete event, or as an IAPI update event.
RA_CREATE_UNMATCHED	If set to true, creates unmatched accounts. If false, don't create the account unless the process rule is set and the workflow it identifies determines that a create is warranted. The default is true.
RA_RESOLVE_PROCESS_RULE	A rule that determines the workflow to run when there are multiple matches using the confirmation rule on the results of the correlation rule.
RA_POPULATE_GLOBAL	Indicates whether to populate the global namespace in addition to the activeSync namespace. The default is false.

Account Attributes

The standard Identity Manager account attributes (defined in the `AttributeDefinition` object) include the following:

- `accountId`
- `email`
- `firstname`
- `fullname`
- `lastname`
- `password`

NOTE These attributes are defined in the Views interface. For more information about these attributes, see *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

Identity Manager account attributes define the Identity Manager attributes of the resource object. The `prototypeXML` section of the adapter file maps the incoming resource attributes to account attributes in Identity Manager. These account attributes are visible when you access the Edit Schema page. (To access the Edit Schema page, click Edit Schema at the bottom of the Edit/Create Resource page.)

The attribute mappings specified in the resource schema map determine which account attributes that can be requested when creating a user. Based on the role selected for a user, the set of account attributes that are prompted for will be the union of attributes of all resources in the selected role. With an Active Sync-enabled adapter, these are the attributes that are available to update the Identity Manager user account. The Active Sync-enabled adapter collects these attributes and stores them in the global area for the input form.

When administrators have created an instance of a resource, they can subsequently use a schema map to:

- Limit resource attributes to only those that are essential for your company.
- Map Identity Manager attributes to resource attributes.
- Create common Identity Manager attribute names to use with multiple resources.
- Identify required user attributes and attribute types.

The schema map is optional. It is provided as a utility to allow the administrator to edit inputs to an Active Sync-enabled adapter, which are often database column names or directory attribute names. Using the schema map and the form, you can implement Java code to handle a resource type, defining details of the resource configuration in the map and form.

For information on creating a resource or editing a resource schema map, see *Sun Java™ System Identity Manager Administration*.

Schema Maps and Active Sync-Enabled Adapters

Identity Manager uses an Active Sync resource's schema map in the same way that it uses a typical schema map. It specifies which attributes to fetch from the resource and their local names. All attribute names that are listed in the schema map (that is, all attributes that exist on the resource) are made available to the Active Sync form and the user form with the `activeSync.name` attribute. If the Active Sync resource does not use a form, all attributes are named `global` to ensure that all attributes automatically propagate to attributes with the same name on all resources. Use a form rather than the global namespace.

TIP	Never put the `accountId` attribute in the global namespace. It is a special attribute that is used for identifying the `waveset.account.global`. The `accountId` attribute also will become a resource's `accountId` directly if the resource account is being created for the first time, bypassing the identity template.

For example, if a new Identity Manager user is created through the Active Sync-enabled adapter and that user has an LDAP account assigned to it, the LDAP `accountID` will match the `global.accountId` instead of the correct DN from the DN template.

How to Map Resource Attributes to Identity Manager Account Attributes

You can map an account attribute to standard Identity Manager account attributes or to a custom attribute (called an *Extended Schema Attribute*).

Code Example 4-2 shows the incoming resource attribute mapped to a standard Identity Manager account attribute. The `<AccountAttributesTypes>` element surrounds the `prototypeXML` section where you map the resource attribute to the Identity Manager account attribute. You define each account attribute with the `<AccountAttributesType>` element.

Code Example 4-2 Defining Account Attributes with <AccountAttributesType>

```
"<AccountAttributeTypes>\n"+
   <AccountAttributeType name='accountId' mapName='change-value-here'
mapType='string' required='true'>\n"+
      "<AttributeDefinitionRef>\nt"+
      <ObjectRef type='AttributeDefinition' name='accountId'/>\n"+
      "</AttributeDefinitionRef>\n"+
   "</AccountAttributeType>\n"+
"</AccountAttributeTypes>\n"+
```

For more information on mapping resource attributes to account attributes, see "Setting Options and Attributes for the Adapter ."

Standard Resource Adapter-Specific Issues with prototypeXML

The following issues are specific to account attributes in resource adapters only:

- User identity template
- Creating an identity template out of multiple user attributes
- Login configuration and pass-through authentication

User Identity Template

The user identity template establishes the account name that is used when creating the account on the resource. It is used to translate information about an Identity Manager user account to account information on the external resource.

Account names are currently in one of two forms:

- Flat namespaces
- Hierarchical namespaces

You can use any schema map attribute (an attribute listed on the left side of the schema map) in the identity template.

You can overwrite the user identity template from the User form. This is commonly done to substitute organization names.

About User Names in Identity Manager

A user has an identity for each of his accounts. This identity can be the same on some or all accounts. The system sets the identity for an account when the account is provisioned. The Identity Manager user object maintains a mapping between a user's identities and the resources to which they correspond. The user has a primary accountId in Identity Manager that is used as a key as well as a separate accountId for each of the resources on which he has an account and is denoted in the form of `accountId:<resource name>`, as shown in Table 4-14.

Table 4-14 Account IDs

Attribute	Example
`accountId`	`maurelius`
`accountId:NT_Res1`	`marcus_aurelius`
`accountId:LDAP_Res1`	`uid=maurelius,ou=marketing,ou=employees,o=abc_company`
`accountId:AIX_Res1`	`maurelius`

Flat Namespaces

You typically use the `accountId` attribute for systems with a flat namespace, which include:

- Microsoft Windows NT 4.0
- UNIX systems (Solaris, AIX, or HP-UX)
- Oracle and Sybase relational databases

For resources with flat namespaces, the identity template can simply specify that the Identity Manager account name should be used.

Hierarchical Namespaces

Distinguished name can include the account name, organizational units, and organizations. It is used for systems with a hierarchical namespace.

For resources with hierarchical namespaces, the template can be more complicated than that of a flat namespace. This allows the full, hierarchical name to be built.

Table 4-15 contains examples of hierarchical namespaces and how they represent distinguished names.

Table 4-15 Hierarchical Namespace Examples

System	Distinguished Name String
LDAP	`cn=$accountId,ou=austin,ou=central,ou=sales,o=comp`
Novell NDS	`cn=$accountId.ou=accounting.o=comp`
Microsoft Windows 2000	`CN=$fullname,CN=Users,DC=mydomain,DC=com`

Here is an example of an identity template for a resource with a hierarchical namespace such as LDAP:

```
uid=$accountID,ou=$department,ou=People,cn=waveset,cn=com
```

Where:

- `accountID` is the Identity Manager account name
- `department` is the user's department name

Creating an Identity Template out of Multiple User Attributes You can also create an identity template out of a portion of multiple user attributes. For example, a template could consist of the first letter of the first name plus seven characters of the last name. In this scenario, you can customize the user form to perform the desired logic and then override the identity template that is defined on the resource.

Login Configuration and Pass-Through Authentication
The `<LoginConfigEntry>` element specifies the name and type of login module as well as the set of authentication properties required by this resource type to complete successful user authentication.

The `<LoginConfig>` and `<SupportedApplications>` sections of the adapter file specify whether the resource will be included in the options list on the Login Module configuration pages. Leave this section of the file unchanged if you want the resource to appear in the options list.

Each `<AuthnProperty>` element contains the following attributes:

Table 4-16 <AuthnProperty> Element Attributes

Attribute	Description
`name`	Identifies the internal authentication property name.
`displayName`	Specifies the value to use when this property is added as an HTML item to the Login form.
`formFieldType`	Specifies the data type that can be either `text` or `password`. This type is used to control whether data input in the HTML field associated with this property is visible (text) or not (password)s
`isId`	Specifies whether this property value should be mapped to the Identity Manager `accountID`. For example, a property should not be mapped when the property value is an X509 certificate.
`dataSource`	Specifies the source for the value of this property. The default value is `user`.
`doNotMap`	Specifies whether to map to a LoginConfigEntry.

Most resource login modules support both the Identity Manager Administrator Interface and the Identity Manager User Interface.

Code Example 4-3 illustrates how `SkeletonResourceAdapter.java` implements the <`LoginConfigEntry`> element:

Code Example 4-3 SkeletonResourceAdapter.java Implementing <LoginConfigEntry>

```
<LoginConfigEntry name='"+Constants.WS_RESOURCE_LOGIN_MODULE+"' type='"+RESOURCE_NAME+"'
displayName='"+RESOURCE_LOGIN_MODULE+"'>\n"+
   "   <AuthnProperties>\n"+
   "      <AuthnProperty name='"+LOGIN_USER+"' displayName='"+DISPLAY_USER+"'
formFieldType='text' isId='true'/>\n"+
   "      <AuthnProperty name='"+LOGIN_PASSWORD+"' displayName='"+DISPLAY_PASSWORD+"'
formFieldType='password'/>\n"+
   "   </AuthnProperties>\n"+
   "   <SupportedApplications>\n"+
   "      <SupportedApplication name='"+Constants.ADMINCONSOLE+"'/>\n"+
   "      <SupportedApplication name='"+Constants.SELFPROVISION+"'/>\n"+
   "   </SupportedApplications>\n"+
"</LoginConfigEntry>\n"+
```

Example LoginConfig Entries

The following example LoginConfig entry is taken from the LDAP resource adapter supplied by Identity Manager. It defines two authentication properties whose `dataSource` value, if not specified, is supplied by the user.

Code Example 4-4 defines the supported Login Module data source options:

Code Example 4-4 Defining Supported Login Module Data Source Options

```
public static final String USER_DATA_SOURCE = "user";
public static final String HTTP_REMOTE_USER_DATA_SOURCE = "http remote user";
public static final String HTTP_ATTRIBUTE_DATA_SOURCE = "http attribute";
public static final String HTTP_REQUEST_DATA_SOURCE = "http request";
public static final String HTTP_HEADER_DATA_SOURCE = "http header";
public static final String HTTPS_X509_CERTIFICATE_DATA_SOURCE = "x509 certificate";
" <LoginConfigEntry name='"+WS_RESOURCE_LOGIN_MODULE+"'
type='"+LDAP_RESOURCE_TYPE+"'
displayName='"+Messages.RES_LOGIN_MOD_LDAP+"'>\n"+
" <AuthnProperties>\n"+
" <AuthnProperty name='"+LDAP_UID+"' displayName='"+Messages.UI_USERID_LABEL+"'
formFieldType='text' isId='true'/>\n"+
" <AuthnProperty name='"+LDAP_PASSWORD+"'
displayName='"+Messages.UI_PWD_LABEL+"'
formFieldType='password'/>\n"+
" </AuthnProperties>\n"+
" </LoginConfigEntry>\n"+
```

Code Example 4-5 shows a Login Config entry where the authentication property's `dataSource` value is not supplied by the user. In this case, it is derived from the HTTP request header.

Code Example 4-5 Login Config Entry

```
" <LoginConfigEntry name='"+Constants.WS_RESOURCE_LOGIN_MODULE+"'
|type='"+RESOURCE_NAME+"'
displayName='"+RESOURCE_LOGIN_MODULE+"'>\n"+
" <AuthnProperties>\n"+
" <AuthnProperty name='"+LOGIN_USER+"' displayName='"+DISPLAY_USER+"'
formFieldType='text'
isId='true' dataSource='http header'/>\n"+
" </AuthnProperties>\n"+|
" </LoginConfigEntry>\n"+
```

Windows NT Object Resource Attribute Declarations

Code Example 4-6 shows how `prototypeXML` defines fields displayed on the Create/Edit Resource page.

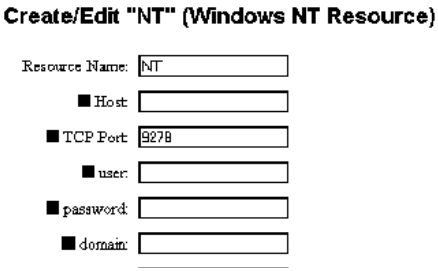
Code Example 4-6 prototypeXML Defining Fields Displayed on Create/Edit Resource Page

```
<ResourceAttributes>
   <ResourceAttribute name='Host' description='The host name running the resource
      agent.' multi='false' value='n'>
   </ResourceAttribute>
   <ResourceAttribute name='TCP Port' description='The TCP/IP port used to communicate
      with the LDAP server.' multi='false' value='9278'>
   </ResourceAttribute>
   <ResourceAttribute name='user' description='The administrator user name with which
      the system should authenticate.' multi='false' value='Administrator'>
   </ResourceAttribute>
   <ResourceAttribute name='password' type='encrypted' description='The password that
      should be used when authenticating.' multi='false' value='VhXrkGkfDKw='>
   </ResourceAttribute>
   <ResourceAttribute name='domain' description='The name of the domain in which
      accounts will be created.' multi='false' value='nt'>
   </ResourceAttribute>
</ResourceAttributes>
```

Identity Manager Rendering of prototypeXML

The Identity Manager Administrator interface displays the resource attributes for the default Windows NT resource as specified in the preceding section.

Figure 4-1 Resource Attributes for Windows NT Resource



Editing the Skeleton File: Overview

The SkeletonActiveSyncResourceAdapter file provides a starting point for creating a new Active Sync-enabled adapter type that administrators can use to create specific instances. Once you have renamed and edited the skeleton file to include the default values you want the adapter to support, you can load it into Identity Manager.

Basic steps include:

- **Name the resource adapter type**. The name you supply here will appear in the New Resources menu in the Identity Manager Administrator Interface.
- **Map resource attributes to Identity Manager account attributes**. This requires that you edit the default values in the skeleton prototypeXML to create your own defaults for this adapter type. For example, you might want to delete the RA_GROUPS attribute that is included in this file from your adapter type.
- **Add or delete methods from the skeleton file**. In particular, add Java code to support the following operations, which are not supported in this example file: joins, leaves, and moves. For example, you want to write an Active Sync-enabled adapter to pull information from an HR database to determine whether employees are still active. To trigger updates based on a change in an employee's active status, you need several items: an effective date column in the database; a window of time to search in; and a list of the updates that have already been done in previous iterations of the search.

Naming the Adapter and Editing Preliminary Information

Editing the preliminary adapter information requires three actions:

- Rename the SkeletonActiveSyncResourceAdapter.java file or sample adapter file to the name of your adapter class.
- Edit code to replace SkeletonActiveSyncResourceAdapter with the new name.

Setting Options and Attributes for the Adapter

The primary way to set options for the adapter type is by mapping the incoming resource attributes to the standard Identity Manager account attributes or creating your own extended schema attributes. The attributes included in SkeletonActiveSyncResourceAdapter are mandatory. Do not delete these attribute definitions when customizing the file.

Mapping a Resource Attribute to a Standard Account Attribute

If you are mapping a resource attribute to one of the standard Identity Manager account attributes, use the syntax shown in Code Example 4-7.

Code Example 4-7 Mapping a Resource Attribute

```
"<AttributeDefinitionRef>\nt"+
   <ObjectRef type='AttributeDefinition' name='accountId'/>\n"+
" </AttributeDefinitionRef>\n"+
```

Use the <AttributeDefinitionRef> element to identify the Identity Manager account attribute. Table 4-17 describes the <AttributeDefinitionRef> element fields.

Table 4-17 <AttributeDefinitionRef> Element Fields

Attribute Field	Description
name	Identifies the Identity Manager account attribute that the resource attribute is being mapped to. (The left column on the resource schema page in the Identity Manager User Interface.)
mapName	Identifies the name of the incoming resource attribute. When editing this skeleton file, replace change-value-here with the resource attribute name.
mapType	Identifies the type of the incoming attribute (string, int, or encrypted).

Mapping Resource Attributes to an Extended Schema Attribute

If you want to map an incoming resource attribute to an attribute other than the standard Identity Manager attributes, you must create an extended schema attribute. Code Example 4-8 illustrates how to map a resource attribute to a custom account attribute.

Code Example 4-8 Mapping Resource Attribute to Custom Account Attribute

```
<AccountAttributeType name='HomeDirectory' type='string'
   mapName='HomeDirectory' mapType='string'>\n"+
</AccountAttributeType>\n"+
```

Note that you do not need to declare an `ObjectRef` type. The `mapName` field identifies the custom account attribute `HomeDirectory`. You can define the `mapType` field the same as you would when mapping an attribute to a standard account attribute.

To test the validity of the resources you just defined (in particular the connection to the resource), save the adapter, load it into Identity Manager, and click Start on the Resources page. The Start button is enabled only if the resource start-up type is Automatic or Manual. The adapter's `init()` and `poll()` methods will be called when the adapter is scheduled.

Setting the Distinguished Name

When creating a new resource, Identity Manager will ask the resource adapter to create a prototype resource that is the XML object definition of the resource.

The distinguished name displays on these Identity Manager user interface pages:

- Resources
- Distinguished Name Template
- Edit Schema

The resource does not need to present the prototype object, it simply has to define the resource attributes and set default values for the resource attributes for which it makes sense to have defaults.

Defining Resource Object Classes, Type, Features, and Attributes

Object types uniquely define a specific type of resource. They are defined in the adapter's prototypeXML section.

Defining an Object Type

The XML `<ObjectTypes>` element is a container within the adapter prototypeXML that contains one or more object type definitions to be managed on that resource.

The XML `<ObjectTypes>` element is a container within the `<ObjectTypes>` element that fully describes the resource-specific object to Identity Manager, including information about:

- Specific object classes that the object type consists of (required only for LDAP-compliant directories)
- List of supported features
- List of object type-specific attributes that are available within Identity Manager for editing and searching.

Table 4-18 lists the supported attributes of the <ObjectType> element.

Table 4-18 Supported <ObjectType> Elements

Attribute	Description
name	Defines the name by which this object type will be displayed and referred to within Identity Manager (required).
icon	Specifies the name of the .gif file to be displayed to the left of objects of this type within the interface. You must install this .gif file in idm/applet/images for use by Identity Manager.
container	If true, specifies that this type of resource object can contain other resource objects of the same or other object types.

Example Object Type Definitions

Code Example 4-9 includes example object type definitions:

Code Example 4-9 Example Object Type Definitions

```
static final String prototypeXml ="<Resource name='Skeleton' class=
   'com.waveset.adapter.sample.SkeletonStandardResourceAdapter'
typeString='Skeleton of a resource adapter'
typeDisplayString='"+Messages.RESTYPE_SKELETON+"'>\n"+
   "   <ObjectTypes>\n"+
   "   <ObjectType name='Group' icon='group'>\n"+
… other content defined below will go here …
"    </ObjectType>\n"+
"    <ObjectType name='Role' icon='ldap_role'>\n"+
… other content defined below will go here …
"    </ObjectType>\n"+
"    <ObjectType name='Organization' icon='folder_with_org' container='true'>\n"+
… other content defined below will go here …
"    </ObjectType>\n"+
"  </ObjectTypes>\n"+
```

Object Classes

Object classes are handled differently for LDAP-based resource objects than for other resource objects.

LDAP-Based Resource Objects

LDAP-based resource objects can consist of more than one LDAP object class, where each object class is an extension of its parent object class. However, within LDAP, the complete set of these object classes is viewed and managed as a single object type within LDAP.

To manage this type of resource object within Identity Manager, include the XML element `<ObjectClasses>` within the `<ObjectType>` definition. The `<ObjectClasses>` element allows you to define the set of object classes that is associated with this `<ObjectType>` as well as the relationship of classes to each other.

Non-LDAP-Based Resource Objects

For non-LDAP-based resource objects, you can use the `<ObjectType>` to represent information other than the resource object type name.

In Code Example 4-10, the `primary` attribute defines the object class to be used when creating and updating an object of this type. In this case, `inetorgperson` is the object class that is defined as the primary one because it is a subclass of the other listed object classes. The `operator` attribute specifies whether the list of object classes should be treated as one (logical AND) or treated as unique classes (logical OR) when listing or getting an object of this type. In this case, Identity Manager performs an AND operation on these object classes prior to any list or get requests for this object type.

Code Example 4-10 Using inetorgperson Object Class

```
<ObjectClasses primary='inetorgperson' operator='AND'>\n"+
   <ObjectClass name='person'/>\n"+
   <ObjectClass name='organizationalPerson'/>\n"+
   <ObjectClass name='inetorgperson'/>\n"+
</ObjectClasses>\n"+
```

In Code Example 4-11, all requests to create and/or update resource objects of this type are done using the `groupOfUniqueNames` object class. All list and get requests will query for all objects whose object class is either `groupOfNames` or `groupOfUniqueNames`.

Code Example 4-11 Using groupOfUniqueNames Object Class

```
<ObjectClasses primary='groupOfUniqueNames' operator='OR'>\n"+
   <ObjectClass name='groupOfNames'/>\n"+
   <ObjectClass name='groupOfUniqueNames'/>\n"+
</ObjectClasses>\n"+
```

Only one object class is defined in Code Example 4-12; so all create, update, list, and get operations will be done using object class `organizationalUnit`.

Code Example 4-12 Using organizationalUnit Object Class

```
<ObjectClasses operator='AND'>\n"+
   <ObjectClass name='organizationalUnit'/>\n"+
</ObjectClasses>\n"+
```

Because there is only one object class, you could exclude the `<ObjectClasses>` section. If left out, the object class defaults to the value of the `<ObjectType>` `name` attribute. However, if you want the object type name to differ from the resource object class name, you must include the `<ObjectClasses>` section with the single `<ObjectClass>` entry.

Object Features

The `<ObjectFeatures>` section specifies a list of one or more features supported by this object type, where each object feature is directly tied to the implementation of the associated object type method in the resource adapter.

Each `ObjectFeature` definition must contain the `name` attribute, which specifies a feature name. The `create` and `update` features may also specify the `form` attribute. This attribute defines the resource form that will be used to process create and update features. If the form attribute is not specified, then Identity Manager processes the features with the same form used by all resources of a given type.

Table 4-19 lists the object feature mappings.

Table 4-19 Object Feature Mappings

Object Feature	Method	Supports Form Attribute?
create	createObject	Yes
delete	deleteObject	No
find	listObjects	No
list	listObjects	No
rename	updateObject	No
saveas	createObject	No
update	updateObject	Yes
view	getObject	No

The `<ObjectFeatures>` section in Code Example 4-13 includes all supported object features. Your resource adapter can support all or a subset. The more object features your adapter supports, the richer the object management function within Identity Manager.

Code Example 4-13 <ObjectFeatures> Section Including all Supported Object Features

```
 <ObjectFeatures>\n"+
   <ObjectFeature name='create' form='My Create Position Form'/>
   <ObjectFeature name='update' form='My Update Position Form'/>
<ObjectFeature name='create'/>\n"+
   <ObjectFeature name='delete'/>\n"+
   <ObjectFeature name='rename'/>\n"+
   <ObjectFeature name='saveas'/>\n"+
   <ObjectFeature name='find'/>\n"+
   <ObjectFeature name='list'/>\n"+
   <ObjectFeature name='view'/>\n"+
</ObjectFeatures>\n"+
```

Object Attributes

The <ObjectAttributes> section specifies the set of attributes that will be managed and queried in Identity Manager. The name of each <ObjectAttribute> element should be the native resource attribute name. Unlike user attributes in Identity Manager, no attribute mapping is specified. Only the native attribute names should be used.

Table 4-20 describes attributes that are required for <ObjectAttributes>.

Table 4-20 Required Attributes for <ObjectAttributes>

Attribute	Description
idAttr	The value of this attribute should be the resource object attribute name that uniquely identifies this object within the resource's object namespace (for example, dn, uid)
displayNameAttr	The value of this attribute should be the resource object attribute name whose value is the name you want displayed when objects of this type are viewed within Identity Manager (for example, cn, samAccountName).
descriptionAttr	(Optional) This value of this attribute should be the resource object attribute name whose value you want displayed in the Description column of the Resources page.

Code Example 4-14 shows an <ObjectAttributes> section defined in an <ObjectType>.

Code Example 4-14 <ObjectAttributes> Section Defined in an <ObjectType>

```
<ObjectAttributes idAttr='dn' displayNameAttr='cn' descriptionAttr=
       'description'>\n"+
   <ObjectAttribute name='cn' type='string'/>\n"+
   <ObjectAttribute name='description' type='string'/>\n"+
   <ObjectAttribute name='owner' type='distinguishedname' namingAttr=
       'cn'/>\n"+
   <ObjectAttribute name='uniqueMember' type='dn' namingAttr='cn' />\n"+
</ObjectAttributes>\n"+
```

Each <ObjectAttribute> has the attributes described in Table 4-21:

Table 4-21 <ObjectAttribute> Attributes

Attribute	Description
name	Identifies the resource object type attribute name (required)
type	Identifies the type of object. Valid types include string or distinguishedname / 'dn' (defaults to string)
namingAttr	If object type is distinguishedname or dn, this value specifies the attribute whose value should be used to display an instance of this object type referred to by the dn within Identity Manager

NOTE The methods in the resource adapter object type implementation is responsible for coercing all string values into the appropriate type based on the resource attribute name.

Assigning a Form

You can assign an optional form for processing incoming data before storing it in Identity Manager. This *resource form* is a mechanism for transforming the incoming data from the schema map and applying it to the user view. The sample form also performs actions (such as enabling and disabling an account) that are based on specific values of incoming data such as employee status.

Defining Resource Forms

You must provide a resource form named <*resource type*> Create <*object type*> Form for each resource <ObjectType> that supports the Create feature. (For example, *AIX Create Group Form* or *LDAP Create Organizational Unit Form*.)

You must also provide a ResourceForm named <*resource type*> Update <object type> Form for each resource <ObjectType> that supports the Update feature. (For example, *AIX Update Group Form* or *LDAP Update Organizational Unit Form*)

Table 4-22 describes attributes contained in the top-level namespace. (All values are strings unless otherwise specified.)

Table 4-22 Top-Level Namespace Attributes

Attribute	Description
`<objectType>.resourceType`	Identity Manager resource type name (for example, LDAP, Active Directory)
`<objectType>.resourceName`	Identity Manager resource name
`<objectType>.resourceId`	Identity Manager resource ID
`<objectType>.objectType`	Resource-specific object type (for example, `Group`)
`<objectType>.objectName`	Name of resource object (for example, `cn` or `samAccountName`)
`<objectType>.objectId`	Fully qualified name of resource object (for example, `dn`)
`<objectType>.requestor`	ID of user requesting view
`<objectType>.attributes`	Resource object attribute name/value pairs (object)
`<objectType>.organization`	Identity Manager member organization
`<objectType>.attrsToGet`	List of object type specific attributes to return when requesting an object through `checkoutView` or `getView` (list)
`<objectType>.searchContex`	Context used to search for non-fully qualified names in form input
`<objectType>.searchAttributes`	List of resource object type-specific attribute names that will be used to search within the specified `searchContext` for names input to the form (list).
`<objectType>.searchTimeLimit`	Maximum time spent searching where `<objectType>` is the lowercase name of a resource specific object type. For example, group, organizationalunit, organization.
`<objectType>.attributes <resource attribute name>`	Used to get or set the value of specified resource attribute (for example, `<objectType>.attributes.cn`, where `cn` is the resource attribute name). When resource attributes are distinguished names, the name returned when getting the value is the value of the `namingAttr` specified in the `<ObjectAttribute>` section of the `<ObjectType>` description.

The next section references forms that show how to access the resource object namespace.

Writing Adapter Methods

The Identity Manager adapter interface provides general methods that you must customize to suit your particular environment. This section briefly describes these methods and covers:

- **Standard resource adapter-specific methods**. These methods are specific to the resource you are updating to synchronize with Identity Manager.
- **Active Sync-specific methods**. These methods provide the mechanism for updating Identity Manager based on pulling information from the authoritative resource. They also include methods for stopping, starting, and scheduling the adapter.

Standard Resource Adapter-Specific Methods

The body of a resource adapter consists of resource-specific methods. Consequently, the resource adapter provides methods that are only generic placeholders for the specific methods that you will write.

This section describes how the methods used to implement operations are categorized. The information is organized into the following sections:

- Creating the Prototype Resource
- Connecting with the Resource
- Checking Connections and Operations
- Defining Features
- Creating Accounts on the Resource
- Deleting Accounts on the Resource
- Updating Accounts on the Resource
- Getting User Information
- List Methods
- Enable and Disable Methods

NOTE When writing a custom adapter, call the `setdisabled()` method on any `WSUser` object that is returned by a custom method.

Creating the Prototype Resource

Table 4-23 describes the methods used to create a Resource instance.

Table 4-23 Methods Used to Create a Resource Instance

Method	Description
`staticCreatePrototypeResource`	Creates a Resource instance from the predefined prototype XML string defined in the resource adapter. Because it is a static method, it can be called knowing only the path to the Java class which is the resource adapter.
`createPrototypeResource`	A local method that can be executed only if you already have an instance of a Java object of the resource adapter class. Typically, the implementation of `createPrototypeResource()` is to just call the `staticCreatePrototypeResource()` method.

Connecting with the Resource

The following methods are responsible for establishing connections and disconnections as an authorized user. All resource adapters must implement these methods.

- `startConnection`
- `stopConnection`

Checking Connections and Operations

`ResourceAdapterBase` provides methods that you can use to check the validity of an operation, such as whether the connection to the resource is working, before the adapter attempts the actual operation.

The methods described in Table 4-24 verify that communication can be made and that the authorized account has access.

Table 4-24 Methods Used to Check Communication

Method	Description
`checkCreateAccount`	Checks to see if an account can be created on the resource. The following features can be checked: • Can basic connectivity to the resource be established? • Does the account already exist? • Do the account attribute values comply with all (if any) resource-specific restrictions or policies that have not been checked at a higher level? This method does not check whether the account already exists. It contains the account attribute information needed to create the user account such as account name, password, and user name. After confirming that the account can be created, the method closes the connection to the resource.
`checkUpdateAccount`	Establishes a connection and checks to see if the account can be updated. This method receives a user object as input. It contains the account attribute information needed to create the user account such as account name, password, and user name. The user object specifies the account attributes that have been added or modified. Only these attributes are checked.
`checkDeleteAccount`	Checks to see if an account exists and can deleted. The following features can be checked: • Can basic connectivity to the resource be established? • Does the account already exist? • Do the account attribute values comply with all (if any) resource-specific restrictions or policies that haven't been checked at a higher level? This method does not check whether the account already exists. It receives a user object as input. It contains the account attribute information needed to delete the user account such as account name, password, and user name. After checking to see if the account can be deleted, the method closes the connection to the resource

Defining Features

The `getFeatures()` method specifies which features are supported by an adapter. The features can be categorized as follows:

- General features
- Account features
- Group features
- Organizational unit features

The `ResourceAdapterBase` class defines a base implementation of the `getFeatures()` method. The Enabled in Base? column in the following tables indicates whether the feature is defined as enabled in the base implementation in `ResourceAdapterBase`.

Table 4-25 General Features

Feature Name	Enabled in Base?	Comments
`ACTIONS`	No	Indicates whether before and after actions are supported. To enable, override the `supportsActions` method with a `true` value.
`RESOURCE_PASSWORD_CHANGE`	No	Indicates whether the resource adapter supports password changes. To enable, override the `supportsResourceAccount` method.

Table 4-26 Account Features

Feature Name	Enabled in Base?	Comments
`ACCOUNT_CASE_ INSENSITIVE_IDS`	Yes	Indicates whether user account names are case-insensitive. Override the `supportsCaseInsensitiveAccountIds` method with a `false` value to make account IDs case-sensitive.
`ACCOUNT_CREATE`	Yes	Indicates whether accounts can be created. Use the remove operation to disable this feature.
`ACCOUNT_DELETE`	Yes	Indicates whether accounts can be deleted. Use the remove operation to disable this feature.
`ACCOUNT_DISABLE`	No	Indicates whether accounts can be disabled on the resource. Override the `supportsAccountDisable` method with a `true` value to enable this feature.

Table 4-26 Account Features *(Continued)*

Feature Name	Enabled in Base?	Comments
ACCOUNT_EXCLUDE	No	Determines whether administrative accounts can be excluded from Identity Manager. Override the `supportsExcludedAccounts` method with a `true` value to enable this feature.
ACCOUNT_ENABLE	No	Indicates whether accounts can be enabled on the resource. Override the `supportsAccountDisable` method with a `true` value if accounts can be enabled on the resource.
ACCOUNT_EXPIRE_ PASSWORD	Yes	Enabled if the expirePassword Identity Manager User attribute is present in the schema map for the adapter. Use the remove operation to disable this feature.
ACCOUNT_GUID	No	If a GUID is present on the resource, use the put operation to enable this feature.
ACCOUNT_ITERATOR	Yes	Indicates whether the adapter uses an account iterator. Use the remove operation to disable this feature.
ACCOUNT_LIST	Yes	Indicates whether the adapter can list accounts. Use the remove operation to disable this feature.
ACCOUNT_LOGIN	Yes	Indicates whether a user can login to an account. Use the remove operation if logins can be disabled.
ACCOUNT_PASSWORD	Yes	Indicates whether an account requires a password. Use the remove operation if passwords can be disabled.
ACCOUNT_RENAME	No	Indicates whether an account can be renamed. Use the put operation to enable this feature.
ACCOUNT_REPORTS_DISABLED	No	Indicates whether the resource reports if an account is disabled. Use the put operation to enable this feature.
ACCOUNT_UNLOCK	No	Indicates whether an account can be unlocked. Use the put operation if accounts can be unlocked.
ACCOUNT_UPDATE	Yes	Indicates whether an account can be modified. Use the remove operation if accounts cannot be updated.
ACCOUNT_USER_PASSWORD_ON_CHANGE	No	Indicates whether the user's current password must be specified when changing the password. Use the put operations if the user's current password is required.

Table 4-27 Group Features

Feature Name	Enabled in Base?	Comments
GROUP_CREATE, GROUP_DELETE GROUP_UPDATE	No	Indicates whether groups can be created, deleted, or updated. Use the put operation to if these features are supported on the resource.

Table 4-28 Organizational Unit Features

Feature Name	Enabled in Base?	Comments
ORGUNIT_CREATE, ORGUNIT_DELETE ORGUNIT_UPDATE	No	Indicates whether organizational units can be created, deleted, or updated. Use the put operation to if these features are supported on the resource.

If your custom adapter overrides the `ResourceAdapterBase` implementation of the `getFeatures` method, add code similar to the following:

```
public GenericObject getFeatures() {
GenericObject genObj = super.getFeatures();
genObj.put(Features.ACCOUNT_RENAME, Features.ACCOUNT_RENAME);
genObj.remove(Features.ACCOUNT_UPDATE, Features.ACCOUNT_UPDATE);
.. other features supported by this Resource Adapter …
return genObj;
}
```

To disable a feature by overriding a different method (such as `supportsActions`) add code similar to the following:

```
public boolean supportsActions() {
   return true;
}
```

The following tables describe the methods used to create, delete, and update accounts on Resources.

Table 4-29 Creating Accounts on the Resource

Method	Description
`realCreate()`	Creates the account on the resource. Receives a user object as input, and contains the account attribute information needed to create the user account (such as account name, password, and user name)

Table 4-30 Deleting Accounts on the Resource

Method	Description
`realDelete()`	Deletes an account(s) on the resource. Receives a user object or list of user objects as input. By default, this method creates a connection, calls `realDelete`, closes the connection for each user object in the list.

Table 4-31 Updating Accounts on the Resource

Method	Description
`realUpdate()`	Updates a subset of the account attributes. By default, this method creates a connection, calls `realUpdate`, and closes the connection for each user object in the list.

NOTE User account attributes from the resource are merged with any new changes from Identity Manager.

Table 4-32 Getting User Information

Method	Description
`getUser()`	Retrieves information from the resource about user attributes. Receives a user object as input (typically with only an account identity set), and returns a new user object with values set for any attribute defined in resource schema map.

You can use list methods to establish processes that adapters use to retrieve user information from the resource.

Table 4-33 List Methods

Method	Description
`getAccountIterator()`	Used to discover or import all the users from a resource. Implements the Account Iterator interface to iterate over all users of a resource.
`listAllObjects ()`	Given a resource object type (such as `accountID` or group), returns a list of that type from the resource. Implement this method to generate lists that are used by the resource, such as a list of resource groups or distribution lists. This method is called from the user form (not called by provisioning engine).

Code Example 4-15 contains code for retrieving information on a resource and converting it to information that Identity Manager can work with.

Code Example 4-15 Resource Adapters: Retrieving Information on a Resource

```
public WSUser getUser(WSUser user)
throws WavesetException
String identity = getIdentity(user);
WSUser newUser = null;
try {
startConnection();
Map attributes = fetchUser(user);
if (attributes != null) {
newUser = makeWavesetUser(attributes);
}
} finally {
stopConnection();
}
return newUser;
}
```

Table 4-34 Enable and Disable Methods

Method	Description
`supportsAccountDisable()`	Returns true or false depending on whether the resource supports native account disable.
`realEnable()`	Implements the native calls that are necessary to enable the user account on the resource.
`realDisable()`	Implements the native calls that are necessary to disable the user account on the resource.

Disabling User Accounts

You can disable an account by using the disable utilities supported by the resource or the account disable utility provided by Identity Manager.

NOTE Use native disable utilities whenever possible.

- **Native support for disabling an account**: Certain resources provide a separate flag that, when set, prevents users from logging in. Example utilities include User Manager for NT, Active Directory Users and Computers for Active Directory, and ConsoleOne or Netware Administrator for NDS/Netware. When an account is enabled, the user's original password is still valid. You can determine whether native support for account disable is available on your resource by implementing the `supportsAccountDisable` method.
- **Identity Manager disable utility**: If the resource does not support disabling an account, or supports disable by means of resetting the user's password, the Identity Manager provisioning engine disables the account. You can perform the disable by setting the user account to a randomly generated, non-displayed, non-retained password. When the account is enabled, the system randomly generates a new password, which is displayed in the Identity Manager Administrator Interface or emailed to the user

Enabling Pass-Through Authentication for Resource Types

Use the following general steps to enable pass-through authentication in a resource type:

1. Ensure that the adapter's `getFeatures()` method returns `ResourceAdapter.ACCOUNT_LOGIN` as a supported feature.
    - If your custom adapter overrides the `ResourceAdapterBase` implementation, add the following code:

```
public GenericObject getFeatures() {
GenericObject genObj = super.getFeatures();
genObj.put(Features.ACCOUNT_RENAME, Features.ACCOUNT_RENAME);
.. other features supported by this Resource Adapter …
return genObj;
}
```

- If your custom adapter does not override the `getFeatures()` implementation in the `ResourceAdapterBase` class, it will inherit the `getFeatures()` implementation that is exported for `ACCOUNT_LOGIN` by default.

2. Add the `<LoginConfigEntry>` element to the adapter's prototypeXML.
3. Implement the adapter's `authenticate()` method.

   The `authenticate()` method authenticates the user against the resource by using the authentication property name/value pairs provided in the `loginInfo` map. If authentication succeeds, be sure that the authenticated unique ID is returned in the WavesetResult by adding a result as follows:

   ```
   result.addResult(Constants.AUTHENTICATED_IDENTITY, accountID);
   ```

   If authentication succeeded, but the user's password was expired, then in addition to the identity added above, also add the password expired indicator to the result to be returned. This will ensure that the user will be forced to change their password on at least resource upon next login to Identity Manager.

   ```
   result.addResult(Constants.RESOURCE_PASSWORD_EXPIRED, new
   Boolean(true));
   ```

   If authentication fails (because the username or password is invalid), then:

   ```
   throw new WavesetException("Authentication failed for " + uid + ".");
   ```

Writing Active Sync-Specific Methods

In this section of the adapter, you must supply the methods that accomplish the main task of your adapter — updating Identity Manager. The methods you write are based on the generic methods supplied with the skeleton adapter file.

You will need to edit some of these methods, which are categorized by task. General guidelines for handling these tasks are discussed in the following sections:

- Initializing and Scheduling the Adapter
- Polling the Resource
- Storing and Retrieving Adapter Attributes
- Updating the Identity Manager Repository
- Shutting Down the Adapter

Initializing and Scheduling the Adapter

You initialize and schedule the adapter by implementing the `init()` and `poll()` methods.

The `init()` method is called when the adapter manager loads the adapter. There are two methods for loading the adapter:

- The manager can load the adapter at system startup if the adapter startup type is automatic.
- An administrator loads the adapter by clicking **Start** on the Resources page if the adapter startup type is manual.

In the initialization process, the adapter can perform its own initialization. Typically, this involves initializing logging (with the `ActiveSyncUtil` class), and any adapter-specific initialization such as registering with a resource to receive update events.

If an exception is thrown, the adapter is shut down and unloaded.

Polling the Resource

All of the adapter's work is performed by the `poll()` method. Scheduling the adapter requires setting up a `poll()` method to search for and retrieve changed information on the resource.

This method is the main method of the Active Sync-enabled adapter. The adapter manager calls the `poll()` method to poll the remote resource for changes. The call then converts the changes into IAPI calls and posts them back to a server. This method is called on its own thread and can block for as long as needed.

It should call its `ActiveSyncUtil` instance's `isStopRequested` method and return when true. Check `isStopRequested` as part of the loop condition when looping through changes.

To configure defaults for polling, you can set the polling-related resource attributes in the adapter file. Setting these polling-related attributes provides administrators with a means to later use the Identity Manager interface to set the start time and date for the poll interval and the length of the interval.

Scheduling Parameters

The following scheduling parameters are used in Active Sync-enabled adapters:

- `RA_SCHEDULE_INTERVAL`
- `RA_SCHEDULE_INTERVAL_COUNT`
- `RA_SCHEDULE_START_TIME`
- `RA_SCHEDULE_START_DATE`

See Table 4-12 for a description of these parameters.

Scheduling Parameters in the prototypeXML

The scheduling parameters are present in the string constant ActiveSync. `ACTIVE_SYNC_STD_RES_ATTRS_XML`, along with all other general Active Sync-related resource attributes.

Sample Polling Scenarios

The following table lists some sample polling scenarios.

Table 4-35 Sample Polling Scenarios

Polling Scenario	Parameters
Daily at 2 A.M.	`Interval = day, count =1, start_time=0200`
Four times daily	`Interval=hour, count=6.`
Poll once every two weeks on Thursday at 5 P.M	`Interval = week, count=2, start date = 20020705` (a Thursday)`, time = 17:00.`

Storing and Retrieving Adapter Attributes

Most Active Sync-enabled adapters are also standard adapters, where a single Java class both extends `ResourceAdapterBase` (or `AgentResourceAdapter`) and implements the Active Sync interface.

The attribute retrieval and update should then pass through to the base as shown in Code Example 4-16:

Code Example 4-16 Attribute Retrieval and Update

```
public Object getAttributeValue(String name) throws WavesetException {
   return getResource().getResourceAttributeVal(name);
}
public void setAttributeValue(String name, Object value) throws WavesetException {
   getResource().setResourceAttributeVal(name,value);
```

Updating the Identity Manager Repository

When an update is received, the adapter uses the `IAPI` classes, notably `IAPIFactory` to:

- Collect the changed attributes
- Map the changes to a unique Identity Manager object.
- Update that object with the changed information

Mapping the Changes to the Identity Manager Object

Using the Active Sync event parameter configurator for the resource, `IAPIFactory.getIAPI` constructs an IAPI object, either `IAPIUser` or `IAPIProcess` from a map of changed attributes. If an exclusion rule (`iapi_create`, `iapi_delete`, or `iapi_update`) is configured for the resource, `IAPIFactory` checks if the account is excluded. If a non-null object is created and returned by the Factory, the adapter can modify the IAPI object (for example, by adding a logger), then submits it.

When the object is submitted, the form associated with the resource is expanded with the object view before the view is checked in. For more information about forms and views, see *Identity Manager Workflows, Forms, and Views*.

In `SkeletonActiveSyncResourceAdapter`, this process is handled in the `buildEvent` and `processUpdates` methods.

Shutting Down the Adapter

No system requirements are associated with adapter shutdown. This is an opportunity for system cleanup.

Installing the Custom Adapter

To install a resource adapter you've customized:

1. Load the `NewResourceAdapter.class` file in the Identity Manager installation directory under

   `idm/WEB-INF/classes/com/waveset/adapter/sample`

   (You might have to create this directory.)

2. Copy the `.gif` file to `idm/applet/images`.

   This `.gif` file is the image that displays next to the resource name on the List Resources page, and it should contain an image for your resource that is 18x18 pixels and 72 DPI in size.

3. Add the class to the `resource.adapter` property in `config/waveset.properties`.

4. Stop and restart the application server. (For information about working with application servers, see *Identity Manager Installation*.)

5. Create an HTML help file for your resource. (For examples, see the `idm.jar` provided in the `com/waveset/msgcat/help/resources` directory. In addition, the procedure for including online help with the application is provided in *Identity Manager Workflows, Forms, and Views*.)

6. Install your adapter and associated help file into Identity Manager.

7. Create a resource in Identity Manager using your adapter.

8. Start your resource and verify connectivity.

Maintaining the Custom Adapter

When you install an Identity Manager service pack, you will need to test the custom resource with the new `idmcommon.jar` and `idmformui.jar` files. You may need to make modifications or enhancements to the custom adapters to adapt to changes made in the new release or service pack. Alternatively, your resource adapter may just need to be rebuilt or refreshed in your installation.

See *Identity Manager Administration* for more information on adapter maintenance.

Testing Your Adapter

After writing the adapter, you must test and load it in to Identity Manager. Testing involves testing both from Identity Manager and running unit tests on your own machine.

You can test the validity of the resources you have just defined (in particular the connection to the resource) by saving the adapter, loading it into Identity Manager, and clicking Start on the List Resources page. The Start button is enabled only if the resource startup type is Automatic or Manual.

This section is organized as follows:

Testing a Custom Adapter

One tool for debugging adapters is the log file that all adapters generate. This log file is generated in the in the `$WSHOME/config` directory and is named `WSTrace1.log`.

NOTE	Tracing must be enabled and the methods for which tracing is requested must be identified for any logging to occur. You can enable tracing through the Identity Manager Debug page or through the command line utility. Also, the custom adapter must include the calls that will make the log entries for the new methods.

The adapter writes to the log file all resource settings that you can use to validate both that the adapter started and that setting changes have been saved.

Active Sync-enabled adapters that make log calls to the ActiveSyncUtil instance will create a log file (or set of log files) in the directory specified by the `Log File Path` resource attribute. Be sure to check these log files for additional Active Sync-related log entries.

General Steps

Follow these general steps when debugging your adapter.

1. Create a test program for your adapter.

   This Java file should perform the following basic functions:

   - Create a new resource
   - Create a user
   - Get a user
   - Update a user
   - Delete a user
   - Perform create, get, update and delete operations on multiple users

   A sample test file (`SkeletonResourceTests.java`) is included in `/REF` on your installation CD.

2. Set your log level as appropriate for your level of debugging. Increase the log level to 4 for your first pass at debugging, and set the log file path and size. When you subsequently start the adapter, it writes to the log file all resource settings that you can use to validate both that the adapter started and that setting changes have been saved.

3. Compile and test your adapter. To compile the test program, use `javac -d . test/filename.java`. (This command creates the class file in the appropriate `com/waveset/adapter/test` directory.) To test your new adapter using this file, make sure that your compiled adapter is in the `com/waveset/adapter` directory and run it using the following:

```
java –D waveset.home=<path> com.waveset.adapter.test.MyResourceAdapter
```

4. Create an HTML help file for your resource. (See `idm.jar` in `com/waveset/msgcat/help/resources` directory for examples. Procedures describing how to include online help with the application, consult *Identity Manager Workflows, Forms, and Views*.)

5. (Active Sync-enabled adapter only) To reset synchronization on the last resource, delete the `XmlData SYNC_`*resourceName* object.

6. Read the error log and modify the adapter.

7. Set the debug level to 2. Level 2 debugging yields information about the adapter settings and any errors, but limits the amount of log detail to a manageable level.

8. Before starting Identity Manager, you must identify the new adapter in the `$WSHOME/config/waveset.properties` file by placing the adapter name under the `resource.adapters` entry — otherwise Identity Manager will not recognize the adapter.

9. Install your adapter and its associated help file into Identity Manager.

NOTE Before an instance of the new adapter can be recognized in the display, you must create a new resource of that type. You can do this through the List Resource page. From this page, select New > *new adapter*. Use the Resource Wizard to create a new adapter.

10. Use Identity Manager to create a resource and a user on that resource.

TIP When debugging an Active Sync-enabled adapter, if you edit the `XmlData SYNC_`*resourceName* object to remove the `MapEntry` for the ActiveSync synchronization process from the Debug page, the adapter starts over from the first detected change.

If you used the IAPI event, set the `Property()` method to store synchronization state for the resource, such as a `last change processed` value. Setting this method is very useful for debugging adapters. You can set the adapter to run and ignore past changes. Subsequently, you can modify the adapter and see the results of your changes in the adapter log file.

If your resource is an Active Sync resource, you may get additional information by turning on logging on the resource edit page. Set the logging level (0-4) and the file path where the log file will be written (as *resource_name*`.log`).

11. (Active Sync-enabled adapter only) Restart synchronization for the last resource.

Tracing Methods in Your Test Adapter

To trace methods in your test adapter

1. Turn on tracing from the Debug page (`debug/Show_Trace.jsp`).
2. Add your adapter using the following format:

   ```
   com.waveset.adapter.sample.MyResourceAdapter
   ```

3. Set the trace level to 4 for maximum output, or 2 for sufficient output.
4. Indicate whether you want the trace information emitted to standard output or sent to a file.
5. To debug the synchronization process further, configure synchronization logging for your test resource. (Instructions are provided in the Identity Manager online help.)

Testing the Resource Object in Identity Manager

You can test your implementation through the Identity Manager Administrator Interface through the Find Resources and List Resources pages.

- Select Resource > List Resource to confirm the following performance characteristics:

Table 4-36 List Resource Performance Characteristics

Expected Behavior in Interface	If not...
Identity Manager includes your resource type in the drop-down list of possible new resources.	Confirm that you have added it to the `resource.adapters` attribute in the `Waveset.properties` file.
When you open the resource folder, its contents reflect all the `<ObjectType>` elements that are defined in your resource adapter's `<ObjectTypes>` section.	Review the `<ObjectType>` elements in the adapter's prototypeXML.
When you right-click on one of your resource object types, all the supported features specified in your resource adapter's `<ObjectFeatures>` section per `<ObjectType>` is available from the menu.	Go to the Debug page and view or edit the resource in question to ensure that its list of `<ObjectFeatures>` for the `<ObjectType>` in question is correct.
You can create new resources and update existing resource objects.	Verify that your resource adapter code is in `WEB-INF/classes/com/waveset/adapter/sample`
The correct ResourceForms have been loaded for each type of operation.	• Confirm that you have checked in all needed resource forms • Verify that the forms are correctly referenced (including the correct case) in the section for forms in the System configuration object.

- Select Resource > Find Resources to confirm the following performance characteristics:

Table 4-37 Find Resources Performance Characteristics

Expected Behavior in the Interface	If not...
You can set all attributes you expect from the Resources **>** Find Resources page	Check all `<ObjectType>` elements and associated `<ObjectAttribute>` elements.
A find resource request returns the appropriate resource objects	Double-check the query arguments to ensure that the appropriate set of resource objects will match that query. If it still doesn't work, try the same query through another LDAP browser to ensure that it is not a problem with the query.
You can edit and/or delete objects returned from your find request.	Check to ensure that the `<ObjectFeatures>` section of the `<ObjectType>` in question includes the Update feature, which enables editing or the Delete feature, which enables deletion.

Viewing a Resource Object

You can view a resource object from the Debug page. To open the Debug page, enter `http://`*`build_name`*`/idm/debug`.

All resource adapter and Active Sync-enabled adapter classes are based on the existing Identity Manager Resource classes.

To view the resource object:

1. Select Resource from the options menu next to List Objects.
2. Click List Objects. This action displays a list of all resource adapters and Active Sync-enabled adapters.
3. Click View next to the resource object you want to view, or click Edit to edit the resource object.

Typical Errors

Typical errors include:

- Form-related errors
- Missing authentication properties
- No Identity Manager user with matching resource account is found

Form-Related Errors

Typical errors with Active Sync-enabled adapters are form-related. These often result from a required field, such as password or email, not being set.

Form validation errors are printed after the final xml of the view. A typical error resembles:

```
20030414 17:23:57.469: result from submit (blank means no errors):
20030414 17:23:57.509: Validation error: missing required field password
```

All messages are also printed out, and they specify the creation and update times of accounts, adapter errors, and a summary of the schema map data.

Resource adapters store information about the last change they processed in the *`SYNC.resourceName XMLData`* object.

Missing Authentication Properties

If you are missing the required authentication property values, make sure the property name is in the set of names dumped in the trace for the specified data source type.

No Identity Manager User with Matching Resource Account is Found

The resource adapter authentication succeeds, but the exception thrown indicates that no Identity Manager user could be found with a matching resource account ID. Confirm that the resource `accountId` associated with the user is the same as the one returned by your resource adapter's authenticate method.

You can check the Identity Manager user's resource `accountIds` through the Debug page. If there is a mismatch, you will either have to change the content of the name being returned by your `authenticate` method or change your resource's ID template to ensure its generating a resource `accountId` that will match the one being returned by authenticate.

Debugging LoginConfig Changes

To debug LoginConfig-related changes to your adapter, you must

1. Enable trace for select files
2. Test single sign-on pass-through authentication login through Telnet

Enabling Identity Manager Trace

Enable Identity Manager level 1 trace for the following classes:

- `com.waveset.security.authn.WSResourceLoginModule`
- `com.waveset.session.LocalSession`
- `com.waveset.session.SessionFactory`
- `com.waveset.ui.LoginHelper`
- `com.waveset.ui.web.common.ContinueLoginForm`
- `com.waveset.ui.web.common.LoginForm`

Testing Single Sign-On (SSO) Pass-Through Authentication

After you have correctly configured the SSO login module, you can telnet directly to the http port and send an http request to `login.jsp`.

You can paste the following request into your telnet session:

```
HEAD /idm/login.jsp HTTP/1.0
Accept: text/plain,text/html,*/*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Host: LOCALHOST
sm_user: Configurator
```

This request includes an SSO login module that is looking for the HTTP header `sm_user`. After pasting this information in the telnet screen, you should see trace indicating that your user has logged in correctly.

For example, see Code Example 4-17:

Code Example 4-17 Sample Output

```
2003.07.08 14:14:16.837 Thread-7 WSResourceLoginModule#checkForAuthenticatedResourceInfo()
Found authenticated resource accountId, 'Configurator@Netegrity SiteMinder' on Identity
Manager user 'Configurator'. null null 2003.07.08 14:14:16.837 Thread-7
WSResourceLoginModule#checkForAuthenticatedResourceInfo()
Exit null null 2003.07.08 14:14:16.837 Thread-7 WSResourceLoginModule#login()
Exit, return code = true null null 2003.07.08 14:14:16.847
Thread-7 LocalSession#login() Login succeeded via Netegrity SiteMinder null null
2003.07.08 14:14:16.847 Thread-7 LocalSession#login() Overall authentication
succeeded null null 2003.07.08 14:14:16.897 Thread-7 LocalSession#checkIfUserDisabled()
Entry null null 2003.07.08 14:14:16.897 Thread-7 LocalSession#checkIfUserDisabled() Exit
null null 2003.07.08 14:14:16.927 Thread-7 LocalSession#login() Exit null null
```

Chapter 5

Working with Firewalls or Proxy Servers

This chapter describes how Identity Manager uses Uniform Resource Locators (URLs) and how to configure Identity Manager to obtain accurate URL data when firewalls or proxy servers are in place.

Servlet APIs

The Web-based Identity Manager user interface is highly dependent on Uniform Resource Locators (URLs) to specify the location of pages to be retrieved by the Web client.

Identity Manager depends on the Servlet APIs provided by an application server (such as Apache Tomcat, IBM WebSphere, or BEA WebLogic) to determine the fully qualified URL in the current HTTP request so that a valid URL can be placed in the generated HTML and HTTP response.

Some configurations prevent the application server from determining the URL the Web client uses for an HTTP request. Examples include:

- A port-forwarding or Network Address Translation (NAT) firewall placed between the Web client and Web server, or between the Web server and application server
- A proxy server (such as Tivoli Policy Director WebSEAL) placed between the Web client and Web server, or between the Web server and application server

For instances in which the Servlet APIs do not provide accurate URL data from an HTTP request, the correct data can be configured in the `Waveset.properties` file (located in your Identity Manager installation `config` directory).

The following attributes control Identity Manager's Web-based documentation root and whether Identity Manager uses the HTML BASE HREF tag:

- `ui.web.useBaseHref` (Default value: `true`) — Set this attribute to one of the following values:
  - `true` — Identity Manager uses the HTML BASE HREF tag to indicate the root of all relative URL paths
  - `false` — All URLs placed into HTML contain fully qualified paths; including scheme, host, and port
- `ui.web.baseHrefURL` — Set this attribute to a non-empty value to define the BASE HREF used in generated HTML, which overrides the value that is calculated using servlet APIs.

  Overriding this calculated value can be useful when those APIs do not return the whole truth, which occurs when:
  - The application server is behind a firewall using port forwarding or NAT
  - The connector between the application server and Web server does not provide accurate information
  - The application server is front-ended by a proxy server

Chapter 6

Configuring Dictionary Support

This chapter describes how to configure a dictionary policy to help protect passwords from simple dictionary attacks. The information is organized as follows:

About the Dictionary Policy

A dictionary policy enables Identity Manager to check passwords against a word database to ensure that they are protected from a simple dictionary attack. If you use this policy with other policy settings to enforce the length and make-up of passwords, Identity Manager makes it difficult for anyone to use a dictionary to guess passwords that are generated or changed in the system.

This dictionary policy extends the password exclusion list specified using the Must Not Contain Words feature on the Edit Policy page (Configure > Policies > Password Policies).

Configuring the Dictionary Policy

To set up a dictionary policy, you must configure dictionary server support and then load the dictionary, as follows:

1. From the Identity Manager administrator user interface, select Configure > Policies and click the Configure Dictionary button.
2. When the Dictionary Configuration page displays, provide the following database information:
   - **Database Type** — Select the database type (Oracle, DB2, SQLServer, or MySQL) that you will use to store the dictionary.
   - **Host** — Enter the name of the host where the database is running.
   - **User** — Enter the user name to use when connecting to the database.
   - **Password** — Enter the password to use when connecting to the database.
   - **Port** — Enter the port on which the database is listening.
   - **Connection URL** — Enter the URL to use when connecting.

     These template variables are available:

     - **%h**: host
     - **%p**: port
     - **%d**: database name
   - **Driver Class** — Enter the JDBC driver class to use while interacting with the database.
   - **Database Name** — Enter the name of the database where the dictionary will be loaded.
   - **Table Naming Context** — Enter the prefix used to name the dictionary table in the database.
   - **Dictionary Filename** — Enter the name of the file to use when loading the dictionary.
3. Click Test to test the database connection.
4. If the connection test is successful, click Load Words to load the dictionary.

NOTE The load task may take a few minutes to complete.

5. Click Test to ensure that the dictionary was loaded correctly.
6. Click Save to save your changes.

Implementing the Dictionary Policy

To implement the dictionary policy,

1. From the Policies page, click the Password Policy link to edit the password policy.
2. On the Edit Policy page, enable the Check passwords against dictionary words option.
3. Click Save to save your changes.

Once implemented, Identity Manager will check all changed and generated passwords against the dictionary.

Chapter 7

Using SPML 1.0 with Identity Manager Web Services

This chapter describes SPML 1.0 support in Identity Manager and Identity Manager Service Provider Edition; including which features are supported and why, how to configure SPML 1.0 support, and how to extend support in the field.

The information is organized as follows:

NOTE This chapter focuses exclusively on SPML 1.0. Unless noted otherwise, all references to SPML in this chapter indicate the *1.0* version.

You will also find concepts discussed here useful when you read about SPML 2.0 in Chapter 8, "Using SPML 2.0 with Identity Manager Web Services."

Who Should Read This Chapter

Application developers and developers who are responsible for integrating Identity Manager can use the SPML 1.0 classes described in this chapter to format service provisioning request messages and to parse response messages.

Working with Identity Manager Web Service Interfaces

Identity Manager Web services are accessed using SOAP messages for HTTP. Identity Manager supports both versions of the OASIS standard for communication with provisioning systems; the Service Provisioning Markup Language (SPML) — versions1.0 and 2.0.

NOTE You can also access Identity Manager Service Provider Edition (SPE) features through SPML 1.0. (These features are not available with SPML version 2.0.)

The SPE SPML interface is very similar to the regular Identity Manager SPML interface — differences in configuration and operation are noted in this chapter where appropriate.

SPML 1.0 is an OASIS standard for providing an open interface to service provisioning activities. SPML 1.0 also has the support of independent software vendors.

NOTE For best performance when working with the Identity Manager Web Interfaces, use the OpenSPML Toolkit that is bundled with Identity Manager. Using the `openspml.jar` file from the `http://www.openspml.org/` website might cause memory leaks.

NOTE The SPE REF Kit provides an `SpmlUsage.java` file that demonstrates how to use the SPE SPML interface.

Configuring SPML

To expose the SPML interface, you must properly configure the Identity Manager server. You must install and modify specific repository objects and edit the `waveset.properties` file.

Installing and Modifying Repository Objects

Table 7-1 describes the repository objects you must install and modify to configure SPML for Identity Manager.

Table 7-1 Repository Objects Used to Configure SPML

Object	Description
`Configuration:SPML`	Contains the definitions of the SPML schemas supported by the server, and rules for converting between the SPML schema and the internal view model. Each SPML schema typically has an associated form.
SPML Forms	Contains one or more form objects that encapsulate the rules for transforming between the external model defined by an SPML schema, and the internal model defined by an Identity Manager view. There typically is one SPML form for each object class defined in the SPML schema.
`Configuration:User Extended Attributes`	Defines user attributes that can be stored in the Identity Manager repository for access through an SPML filter.
`Configuration:UserUIConfig`	Contains additional *queryable* and summary attributes for Identity Manager user objects. A queryable attribute must be defined for any attribute you want to use in an SPML filter. A summary attribute must be defined for any attribute you want to return in an optimized search
`TaskDefinition:SPMLRequest`	System task used to process asynchronous SPML requests. There should be no need to customize this object.

Identity Manager provides a sample set of SPML configuration objects in the `sample/spml.xml` file. You must import this file manually because it is not imported by default when the repository is initialized.

The sample configuration defines a class named `person` that tracks the evolving standard schema defined by the SPML working group. Though the working group has not yet published a standard user schema, it will almost certainly be based upon the popular LDAP `inetorgperson` schema.

NOTE Avoid customizing the `person` class. Instead, keep it consistent with the standard schema.

NOTE To configure the SPE SPML interface, you must install and modify the `Configuration:SPE SPML` configuration object.

- Configure the `person` class (the only objectclass defined by default) to use the SPE-specific view handler (`IDMXUser`).
- Use the `form` attribute to define a user form that translates between the SPML request/response and the view.

  The `form` attribute can take a special value `(view)`: in which no form processing is applied to the view. (For example, the `view` is passed directly between the client and Identity Manager.)

Use the following (default) path to access the SPE SPML interface:

```
/servlet/spespml
```

For example, if you deploy Identity Manager in the `/idm` context on `localhost` and on port 8080, you can access the interface at the following URL:

`http://localhost:8080/idm/servlet/spespml`

Editing the waveset.properties File

Table 7-2 describes three optional entries in the `waveset.properties` file that you can use to control how SPML requests are authorized.

Table 7-2 Optional Entries in waveset.properties

Entry Name	Description
`soap.username`	The name of an Identity Manager user that is to be used as the effective user for performing SPML requests
`soap.password`	The clear text password for the user specified by `soap.username`
`soap.epassword`	The base-64 representation of the encrypted password for the user specified by `soap.username`

Editing the soap.epassword and soap.password Properties

The user specified with `soap.username` is referred to as the *proxy user*. If you want to define a proxy user, you must set `soap.username`, but only one of the two password entries must be set. Using `soap.password` is the simplest, but this exposes a clear text password in the properties file. Using `soap.epassword` is more secure, but requires extra steps to generate the encrypted password.

Establishing a proxy user is convenient for clients because they are not required to authenticate to use the Web service. This is a common configuration for portal environments where the Identity Manager server is accessed only by another application that itself handles authentication of users.

CAUTION This is a dangerous configuration if the HTTP port on which the server is responding is generally accessible. Anyone who knows the URL of the Identity Manager server and understands how to build SPML requests can perform any Identity Manager operation that the proxy user could perform.

The SPML standard does not specify how authentication and authorization are performed. There are several related Web standards for authentication but these will not be in widespread use for some time. Probably, the most common near-term approach for authentication will be to rely on the use of SSL between applications and the server. How SSL is configured cannot be dictated by Identity Manager.

In cases where neither a proxy user or SSL can be used, Identity Manager does support a vendor-specific extension to SPML that allows the client to log in and maintain a *session token* that is used to authenticate subsequent requests. The easiest way to do this is to use the `LighthouseClient` class, an extension of the `SpmlClient` class that provides support for specifying credentials, performing a login request, and passing a session token in all SPML requests.

NOTE The SPE SPML interface does not support authentication and authorization. SPE assumes that the client accessing Identity Manager has already been authenticated and authorized by an access management application. The client has all possible rights when using the SPE SPML interface.

To prevent sensitive data from being exposed between the client and Identity Manager, consider accessing the SPE SPML interface over SSL.

Obtaining an Encrypted Password

One way to obtain an encrypted password is to use the `encrypt` command in the Identity Manager console. Another way is to view the XML for the proxy user in the Debug pages or from the console. Look in the `WSUser` element for the value of the `password` attribute. You can use this value as the value for the `soap.epassword` property.

Editing Configuration Objects

Applications require a mechanism to send SPML messages and receive SPML responses.

To configure SPML for Identity Manager, you must work with the following configuration objects:

- `Configuration:SPML`
- `Configuration:User Extended Attributes`
- `Configuration:UserUIConfig`
- `TaskDefinition:SPMLRequest`
- `SPML Forms`

NOTE The SPE SPML interface has only one configuration object, `Configuration:SPE SPML`, which is similar to the `Configuration:SPML` object in structure.

Editing the Configuration: SPML

The SPML object contains definitions for the SPML schemas you want to expose and information about how those SPML schemas are mapped into Identity Manager views. This information is represented using a `GenericObject` that is stored as an extension of the Configuration object.

There are two attributes defined in this `GenericObject`: `schemas` and `classes`.

- **Schemas**: A list of strings, where each string contains the escaped XML for one SPML `<schema>` element. Because the SPML elements are not defined in `waveset.dtd`, you cannot directly include them in an Identity Manager XML document. Instead, you must include them as escaped text.

- **Classes**: A list of objects containing information about the supported SPML classes and how they are mapped onto views. There should be one object on this list for each class defined by the SPML schemas on the `schemas` list.

The distinction between the two lists can be confusing at first. The information about the **schemas** list defines what Identity Manager will return in response to an SPML SchemaRequest message. This information can be used by the client to understand which attributes can be included in other messages such as `AddRequest`. Identity Manager does not care about the contents of the schemas list. This list is simply returned verbatim to the client.

You are not required to define any SPML schemas. Identity Manager works without schemas. If no SPML schema has been defined, Identity Manager returns an empty response when it receives a schema request message. Without a schema, clients must rely on pre-existing knowledge about the supported classes and attributes. While this is often the case, it is still considered good practice to write SPML schemas, so that general purpose tools such as the OpenSPML Browser can be used to build requests.

Default SPML Configuration

Code Example 7-1 shows the default SPML configuration. The text of the SPML schema definitions have been omitted for brevity.

Code Example 7-1 Default SPML Configuration

```
<Configuration name='SPML'>
   <Extension>
      <Object>
         <Attribute name='classes'>
            <List>
               <Object name='person'>
                  <Attribute name='type' value='User'/>
                  <Attribute name='form' value='SPMLPerson'/>
                  <Attribute name='default' value='true'/>
                  <Attribute name='identifier' value='uid'/>
               </Object>
               <Object name='request'>
                  <Attribute name='type' value='TaskInstance'/>
                  <Attribute name='filter'>
                     <AttributeCondition attrName='defName' operator='equals'
                      operand='SPMLRequest'/>
                  </Attribute>
               </Object>
            </List>
         </Attribute>
         <Attribute name='schemas'>
            <List>
               <String>
```

Code Example 7-1 Default SPML Configuration *(Continued)*

```
                  <![CDATA[
                  <schema xmlns="urn:oasis:names:tc:SPML:1:0"
                  ...SPML standard schema...
                  </schema>
                  ]]>
               </String>
               <String>
                  <![CDATA[
                  <schema xmlns="urn:oasis:names:tc:SPML:1:0"
                  ...Waveset custom schema...
                  </schema>
                  ]]>
               </String>
            </List>
         </Attribute>
      </Object>
   </Extension>
</Configuration>
```

Two classes are defined in this example: the standard `person` and an Identity Manager extension named `request`. The following attributes are supported in a class definition:

- **name**: Identifies the name of the class. This value can correspond to an `<ObjectClassDefinition>` element in an SPML schema, although this is not required. This name is used as the value of the `objectclass` attribute in an Add request or Search request.
- **type**: Defines the Identity Manager view type used to manage instances of this class. This is usually `User`, but can be any repository type that can be accessed through a view. For information about views, see *Sun Java™ System Identity Manager Workflows, Forms, and Views*.
- **form**: Identifies the name of a configuration object containing a form. This form contains the rules for transforming between the external attributes defined by the class and the internal view attributes.
- **default**: When set to `true`, indicates that this is the default class for this type only. If you have more than one SPML class implemented on the same type, one should be designated as the default.
- **identifier**: Each class typically defines one attribute that is considered to be the identity of the object. Where possible, the value of this attribute is used as the name of the corresponding repository object that you create to represent the instance. The *identifier* attribute in the class definition specifies which attribute represents the identity.

- **filter**: When an SPML search request is evaluated for a class, you typically include all repository objects associated with that class in that search. This is fine for User objects, but some classes may be implemented using generic types such as `TaskDefinition` or `Configuration`, not all of which are considered instances of the SPML class.

  To prevent unwanted objects from being included in the search, you can specify the filter attribute. The value is expected to be an `<AttributeCondition>` element or a `<List>` of `<AttributeCondition>` elements. Because custom classes are almost always made for the User type, using a filter is uncommon. The default configuration uses them to expose a subset of the `TaskInstance` objects that are known to have been created to handle asynchronous SPML requests.

Default Schemas

The `schemas` attribute contains a list of strings that contain the escaped XML for an SPML `<schema>` element. If you examine the `spml.xml` file, you will notice the schema elements are surrounded with a CDATA marked section, which is more convenient for escaping long strings of XML. When this is normalized, it will be converted into a string containing `&lt;` character entities.

The default configuration includes two schemas:

- Standard schema being defined by the SPML working group
- Custom schema defined by Identity Manager. Do not customize these schemas.The Identity Manager schema contains a class definition for **request** as well as a number of extended requests for common account management operations.

Editing the Configuration: SPMLPerson Object

Each class defined in `Configuration:SPML` typically has an associated form object that contains the rules for transforming between the external attribute model defined by the class and the internal model defined by the associated view.

The standard `person` class references the following form (Code Example 7-2):

Code Example 7-2 Standard Person Class References Form

```
<Configuration name='SPMLPerson'>
   <Extension>
      <Form>
         <Field name='cn'>
            <Derivation><ref>global.fullname</ref></Derivation>
         </Field>
         <Field name='global.fullname'>
            <Expansion><ref>cn</ref></Expansion>
         </Field>
         <Field name='email'>
            <Derivation><ref>global.email</ref></Derivation>
         </Field>
         <Field name='global.email'>
            <Expansion><ref>email</ref></Expansion>
         </Field>
         <Field name='description'>
            <Derivation>
               <ref>accounts[Lighthouse].description</ref>
            </Derivation>
         </Field>
         <Field name='accounts[Lighthouse].description'>
            <Expansion><ref>description</ref></Expansion>
         </Field>
         <Field name='password'>
            <Derivation><ref>password.password</ref></Derivation>
         </Field>
         <Field name='password.password'>
            <Expansion><ref>password</ref></Expansion>
         </Field>
         <Field name='sn'>
            <Derivation><ref>global.lastname</ref></Derivation>
         </Field>
         <Field name='global.lastname'>
            <Expansion><ref>sn</ref></Expansion>
         </Field>
```

Code Example 7-2 Standard Person Class References Form *(Continued)*

```
          <Field name='gn'>
             <Derivation><ref>global.firstname</ref></Derivation>
          </Field>
          <Field name='global.firstname'>
             <Expansion><ref>gn</ref></Expansion>
          </Field>
          <Field name='telephone'>
             <Derivation>
                <ref>accounts[Lighthouse].telephone</ref>
             </Derivation>
          </Field>
          <Field name='accounts[Lighthouse].telephone'>
             <Expansion><ref>telephone</ref></Expansion>
          </Field>
       </Form>
   </Extension>
</Configuration>
```

NOTE SPML class forms contain no `<Display>` elements. These forms are defined only for data transformation, not for interactive editing.

For every attribute in a class definition, there is a pair of field definitions. One field uses a `<Derivation>` expression to transform the internal view attribute `name` to the external name. One field uses an `<Expansion>` expression to transform the external `name` to the internal `name`.

The form is processed in such a way that when attributes are returned to the client, only the result of the `<Derivation>` expressions are included. When attributes are being sent from the client to the server, only the results of the `<Expansion>` expressions are assimilated back into the view. The effect is similar to the schema map of a Resource definition.

Editing the Configuration: User Extended Attributes Object

Any attributes that you want to use in an SPML search filter must be defined as an *extended attribute* for Identity Manager users. This causes the value of the attribute to be stored in the Identity Manager repository, even if that value is also stored as a resource account attribute. You should generally try to minimize the number of extended attributes because it increases repository size as well as the chances of consistency problems between attributes stored in Identity Manager and the real value of the attribute stored on a resource. But for an attribute to be used in an Identity Manager query, it must be declared as extended so that the value is always accessible when the repository query indexes are built.

You must define as an *extended attribute* any attribute that you want to include in the set of *summary attributes* for the user. You can use summary attributes to optimize searches by avoiding deserialization of the object XML, and instead return only a few of the most important attributes of the user. In the Identity Manager SPML implementation, summary attributes are returned whenever you do not explicitly provide a list of return attributes in the search request.

In the default SPML configuration, the attributes `telephone` and `description` from the standard `person` schema are declared as extended attributes.

Code Example 7-3 telephone and description Declared as Extended Attributes

```
<Configuration id='#ID#Configuration:UserExtendedAttributes' name='User Extended
Attributes'>
   <Extension>
      <List>
         <!-- this is the standard set -->
         <String>firstname</String>
         <String>lastname</String>
         <String>fullname</String>
         <!-- these are the SPML extensions -->
         <String>description</String>
         <String>telephone</String>
      </List>
   </Extension>
</Configuration>
```

You can customize the list of attributes according to the needs of your site.

The names you choose for the extended attributes depend on the mappings performed in the class form. Because the default SPMLPerson form maps `sn` into `lastname`, the extended attribute must be declared as `lastname`. Because the form does not transform the name of `telephone` or `description`, the extended attribute name comes directly from the SPML schema.

Beyond declaring extended attributes, you must also modify the `Configuration:UserUIConfig` object to declare which of the attributes are to be queryable (that is, usable in an SPML filter) and which are to be summary attributes (returned by an optimized search result).

Editing the Configuration: UserUIConfig Object

You must declare:

- Any attributes you want to use in an SPML search filter in the `<QueryableAttrNames>` section of the `UserUIConfig` object.
- Any attributes you want to return in an optimized SPML search result in the `<SummaryAttrNames>` section.

Code Example 7-4 defines the extended attribute `telephone` as a queryable and summary attribute. It also has declarations for `firstname` and `lastname` but those are usually already declared. You can customize these lists according to the needs of your site.

Code Example 7-4 telephone Defined as Queryable and Summary Attribute

```
<Object>
   <Attribute name='add'>
      <Object>
         <Attribute name='SummaryAttrNames'>
            <List>
               <!-- these are usually there, but make sure -->
               <String>firstname</String>
               <String>lastname</String>
               <!-- this is an SPML addition -->
               <String>telephone</String>
            </List>
         </Attribute>
         <Attribute name='QueryableAttrNames'>
            <List>
               <!-- these are usually there, but make sure -->
               <String>firstname</String>
               <String>lastname</String>
               <!-- this is an SPML addition -->
               <String>telephone</String>
            </List>
         </Attribute>
```

Code Example 7-4 telephone Defined as Queryable and Summary Attribute *(Continued)*

```
        </Object>
      </Attribute>
   </Object>
```

Editing the TaskDefinition: SPMLRequest Object

The `spml.xml` file also includes a brief definition for a new system task named `SpmlRequest`. This task is used to implement asynchronous SPML requests. When the server receives an asynchronous request, a new instance of this task is launched, and the SPML message is passed as an input variable for the task. The repository ID of the task instance is then returned in the SPML response for later status requests.

```
<TaskDefinition name='SPMLRequest'
   executor='com.waveset.rpc.SpmlExecutor'
   execMode='asyncImmediate'
   resultLimit='86400'>
</TaskDefinition>
```

You must not change the name of the definition, the name of the executor, or the execution mode. You may, however, want to change the value of `resultLimit`. When an asynchronous request has completed, the result is typically retained for a period of time so that the client can issue an SPML status request to obtain the results. How long the results should be retained will be site-specific.

If non-negative, `resultLimit` specifies the time (in seconds) that the system will retain results after a task has completed. The default value for `SPMLRequests` is typically 3600 seconds, or approximately one hour. Other tasks default to 0 seconds unless the task is changed to another value.

If negative, the request instance will never be removed automatically.

TIP Set the value of `resultLimit` to the shortest possible time to avoid cluttering the repository.

NOTE The SPE SPML interface does not support asynchronous requests.

Deployment Descriptor

The Identity Manager deployment descriptor, typically found in the file `WEB-INF/web.xml`, must contain a declaration for the servlet that receives SOAP messages.

If you are having difficulty contacting the SPML web service, look in the `web.xml` file for a servlet declaration that looks like Code Example 7-5:

Code Example 7-5 Servlet Declaration

```
<servlet>
   <servlet-name>rpcrouter2</servlet-name>
   <display-name>OpenSPML SOAP Router</display-name>
   <description>no description</description>
   <servlet-class>
      org.openspml.server.SOAPRouter
   </servlet-class>
   <init-param>
      <param-name>handlers</param-name>
      <param-value>com.waveset.rpc.SimpleRpcHandler</param-value>
   </init-param>
   <init-param>
      <param-name>spmlHandler</param-name>
      <param-value>com.waveset.rpc.SpmlHandler</param-value>
   </init-param>
   <init-param>
      <param-name>rpcHandler</param-name>
      <param-value>com.waveset.rpc.RemoteSessionHandler</param-value>
   </init-param>
</servlet>
```

This declaration allows you to access the `addRequest`, `modifyRequest`, and `searchRequest` web services through the URL:

`http://`*<host>*`:`*<port>*`/idm/servlet/rpcrouter2`

You do not need to define a *<servlet-mapping>* (although you can). Do not modify the contents of this servlet declaration.

Understanding How Requests Are Processed

This section gives a general overview of how SPML requests are processed in Identity Manager.

How an Add Request is Processed

The following steps describe the processing of an Add request.

1. An SPML `<addRequest>` message is received. The request must include a value for the `objectclass` attribute.
2. The server examines the `Configuration:SPML` object to find the definition for the class. From the class definition, it obtains the associated view type and form name.
3. The server calls the `Session.createView` method to construct a new view for that type.
4. The attributes included in the request are processed by the class form. The results of the `<Expansion>` expressions are assimilated into the view.
5. The view is checked in.

How a Modify Request is Processed

The following steps describe how a Modify request is processed:

1. An SPML `<modifyRequest>` message is received. The request can include an optional *objectclass* attribute. The request must contain an identifier for an existing object. The identifier must include both the repository type and the object name.
2. The server calls `Session.checkoutView` for the existing object.
3. The server examines the `Configuration:SPML` object to find the definition for the class. If an `objectclass` attribute was passed in the request, that value determines the class. Otherwise, the class marked as the default class for the repository type is used.
4. The attributes included in the request are processed by the form that is specified by the class definition. The results of the `<Expansion>` expressions are assimilated into the view.
5. The view is checked in.

How a Search Request is Processed

The following steps describe how a Search request is processed:

1. An SPML `<searchRequest>` message is received. The request can include an optional *objectclass* attribute.
2. The server examines the `Configuration:SPML` object to find the definition for the class. If an `objectclass` attribute was passed in the request, that determines the class. Otherwise, the class marked as the default class for the User type is used.
3. If the request includes a filter, it is converted to a list of `AttributeCondition` objects. Because the filter terms are written using the external names, the class form is consulted to convert these into the names of queryable attributes on the repository type.
4. The server calls the `Session.listObjects` method with the repository type and optional conditions.
5. The server builds the search response by iterating over each row of the `listObjects` call applying the following steps.
6. If no list of return attributes is specified in the search, only the *summary attributes* defined for the repository type are returned. The class form is used to convert the internal summary attribute names into the external names.
7. If a list of return attributes is specified, and these all correspond to summary attributes, the summary attribute values are returned. The form is again used to convert internal to external names.
8. If a return attribute is specified that is not a summary attribute, the server calls `Session.getView` on this object to materialize the view. The view is processed with the class form and the results of the `<Derivation>` expressions are captured and returned as the results for that row.

Identity Manager first attempts to satisfy a search request using only the summary attributes defined for a type. The result is a much faster search, especially if no filter is specified and many objects are included in the result. If a view must be instantiated to perform the search, the search will be substantially slower, and should only be performed if a filter is specified that restricts the result to a small number of objects.

If you know the identity of an object and want to retrieve its attributes without writing a filter expression, use the identity as the value of the *baseContext* in the search request. This will result in a faster search, because Identity Manager can avoid the query and just build the view.

If you do not specify return attributes, only the summary attributes are returned. Consequently, if you want all attributes, you must explicitly include them in the return attribute list. Because requesting all available attributes is a common operation, this is somewhat inconvenient. As an alternative to specifying a complete list of attributes, Identity Manager recognizes a single return attribute named `view` to indicate that the object view should be fully instantiated and processed with the class form to produce the result.

You can also specify an attribute level (such as `accounts[Lighthouse]` or `accountinfo`). If you specify a level, Identity Manager will return all attribute values within the scope of that level for the fully instantiated view.

Launching the SPML Browser

You can use the OpenSPML Browser application to test the Identity Manager SPML configuration.

To launch the browser from the command line, type:

```
lh spml
```

NOTE For more information about using the `lh spml` command, see *Sun Java™ System Identity Manager Administration*.

Connecting to the Identity Manager Server

To connect to the Identity Manager server, open the **Connect** page and enter the URL of the Identity Manager server. For example, if your server is running on port 8080 on the local machine, the URL would be:

```
http://localhost:8080/idm/servlet/rpcrouter2
```

Where `localhost` is the machine on which you are running Identity Manager.

Testing and Troubleshooting the SPML Configuration

To test your SPML configuration:

1. Open the **Connect** page and click **Test**.

   A dialog indicating that the connection was successful pops up.

2. Open the **Schema** page and click **Submit**.

   The system displays a tree view of the schemas supported by the Identity Manager server.

If you cannot establish a successful connection

- Double-check the URL you entered.
- If the error you receive contains phrases such as "`no response`" or "`connection refused`," then the problem is most likely the host or port used in the connection URL.
- If the error suggests that a connection was made, but the web application or servlet could not be located, the problem is most likely in the `WEB-INF/web.xml` file. See "Deployment Descriptor" on page 43 for more information.

Developing SPML Applications

Once the server is configured, an application will need a mechanism to send SPML messages and receive SPML responses. For Java applications, the easiest way to accomplish this is to make use of the OpenSPML Toolkit. The toolkit is available from `www.openspml.org` and is bundled with Identity Manager.

The toolkit provides the following components:

- Java class model for SPML messages
- Classes to send and receive messages on the client
- Classes to receive and process requests on the server

Table 7-3 summarizes the most important classes provided by the toolkit. Each request type has a corresponding class. Consult the JavaDocs distributed with the toolkit for complete information.

Table 7-3 Classes Provided by OpenSPML Toolkit

Class	Description
`AddRequest`	Constructs a message that requests the creation of a new object. The type of object to create is defined by passing an attribute named `objectclass`. Other attributes passed should adhere to the schema associated with the object class. SPML does not yet define any standard schemas. Identity Manager can be configured to support any desired schema.
`ModifyRequest`	Constructs a message that requests the modification of an existing object. You need include only those attributes that you want to modify in the request. Attributes not included in the request retain their current value.
`DeleteRequest`	Constructs a message that requests the deletion of an object.
`SearchRequest`	Constructs a message that requests attributes of objects that match certain criteria.
`BatchRequest`	Constructs a message that can contain more than one SPML request.
`CancelRequest`	Constructs a message that cancels a request that was formerly executed asynchronously.
`SchemaRequest`	Constructs a message that requests information about the SPML object classes supported by the server.
`StatusRequest`	Constructs a message that requests the status of a request that was formerly executed asynchronously.
`SpmlResponse`	The base class for objects representing response messages sent back from the server. Each request class has a corresponding response class: `AddResponse` and `ModifyResponse`, for example.
`SpmlClient`	Provides a simple interface for sending and receiving SPML messages.

NOTE The SPE REF Kit provides an `SpmlUsage.java` file that demonstrates how to use the SPE SPML interface. This REF Kit also contains an `ant` script that compiles the `SpmlUsage` class.

Usage:

```
java [ -Dtrace=true ] com.sun.idm.idmx.example.SpmlUsage [ URL ]
```

where `URL` points to the SPE SPML interface and defaults to

```
http://localhost:8080/idm/spespml
```

If you enable trace for SPE, all SPE SPML messages will be printed to the standard output.

ExtendedRequest Examples

Table 7-4 describes the ExtendedRequest classes used to send and receive messages on the client.

Table 7-4 ExtendedRequest Classes for Sending and Receiving Messages

ExtendedRequest	Description
`deleteUser`	Constructs a message that requests the deletion of a user.
`disableUser`	Constructs a message that requests the disabling of a user.
`enableUser`	Constructs a message that requests the enabling of a user.
`resetUserPassword`	Constructs a message that requests the reset of a user password.
`changeUserPassword`	Constructs a message that requests the change of a user password.
`launchProcess`	Constructs a message that requests the launch of a process.
`listResourceobjects`	Constructs a message to request the name of a resource object in the Identity Manager repository, and the type of object supported by that resource. The request returns a list of names.
`runForm`	Allows you to create custom SPML requests that return information obtained by calling the Identity Manager Session API.

The server code converts the extended requests into view operations.

Sample Extended Request

Extended requests typically take the format shown in Code Example 7-6:

Code Example 7-6 Extended Request Format

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("changeUserPassword");
req.setAttribute("accountId", "jlarson");
req.setAttribute("password", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

Most SPML extended requests take the following arguments:

- `accountId` — Identifies the Identity Manager user name
- `accounts` — Lists resource names in a comma-delimited list

If you do not pass an `accounts` attributes, the operation will update all resource accounts linked to the user, including itself. If you do pass `accounts`, the operation updates only the specified resources. You must include `Lighthouse` in a non-null accounts list if you want to update the Identity Manager user in addition to specific resource accounts.

deleteUser

Code Example 7-7 shows a typical format for `deleteUser` request (View — Deprovision view).

Code Example 7-7 deleteUser Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("deleteUser");
req.setAttribute("accountId", "jlarson");
req.setAttribute("accounts", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

disableUser

Code Example 7-8 shows a typical format for `disableUser` request (View — Disable view).

Code Example 7-8 disableUser Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("disableUser");
req.setAttribute("accountId", "jlarson");
req.setAttribute("accounts", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

enableUser

Code Example 7-9 shows a typical format for `enableUser` request (View — Enable view).

Code Example 7-9 enableUser Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("enableUser");
req.setAttribute("accountId", "jlarson");
req.setAttribute("accounts", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

resetUserPassword

Code Example 7-10 shows a typical format for `resetUser` request (View — Reset User Password view).

Code Example 7-10 resetUser Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("resetUserPassword");
req.setAttribute("accountId", "jlarson");
req.setAttribute("accounts", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

changeUserPassword

Code Example 7-11 shows a typical format for `changeUserPassword` request. (View — Change User Password view).

Code Example 7-11 changeUserPassword Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("changeUserPassword");
req.setAttribute("accountId", "jlarson");
req.setAttribute("password", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

launchProcess

Code Example 7-12 shows a typical format for `launchProcess` request. (View — Process view).

Code Example 7-12 launchProcess Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("launchProcess");
req.setAttribute("process", "my custom process");
req.setAttribute("taskName", "my task instance");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

Where:

- **Process** — name of the workflow to launch
- **Workflow** — any TaskName

The `process` attribute names a Task Definition object in the Identity Manager repository that is to be run. The `taskName` attribute is used to name the Task Instance object that is created to hold the runtime state of the process. The remaining attributes are arbitrary, and are passed into the task. The `launchProcess` request can be used to start any custom process.

listResourceObjects

Code Example 7-13 shows a typical format for `listResourceObjects` request.

Code Example 7-13 listResourceObjects Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("listResourceObjects");
req.setAttribute("resource", "LDAP");
req.setAttribute("type", "group");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

Where:

- **resource**: Specifies the name of a Resource object in the Identity Manager repository
- **type**: Specifies the type of an object supported by that resource

runForm

Code Example 7-14 shows a typical format for a `runForm` request.

Code Example 7-14 runForm Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("runForm");
req.setAttribute("form", "SPML Get Object Names");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

Where `form` is the name of a Configuration object containing a form.

Example Form

The form shown in Code Example 7-15 runs queries and returns a list of the Role, Resource, and Organization names accessible to the current user.

Code Example 7-15 Query Form

```
<Configuration name='SPML Get Object Names'>
  <Extension>
    <Form>
      <Field name='roles'>
        <Derivation>
          <invoke name='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>Role</s>
          </invoke>
        </Derivation>
      </Field>
      <Field name='resources'>
        <Derivation>
          <invoke name='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>Resource</s>
          </invoke>
        </Derivation>
      </Field>
```

Code Example 7-15 Query Form *(Continued)*

```
      <Field name='organizations'>
        <Derivation>
          <invoke name='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>ObjectGroup</s>
          </invoke>
        </Derivation>
      </Field>
    </Form>
  </Extension>
</Configuration>
```

The `runForm` request allows you to create custom SPML requests that return information obtained by calling the Identity Manager Session API. For example, when configuring a user interface for editing users, you might want to provide a selector that displays the names of the organizations, roles resources, and policies that can be assigned to a user. You can configure the SPML interface to expose these objects as SPML object classes then use a `searchRequest` to query for their names. However, this would require four `searchRequests` to gather the information. You can reduce the number of SPML requests by instead encoding the queries in a form, then using a single `runForm` request to perform the queries and return the combined results.

Using Trace with SPML

SPML provides the following options for turning on trace output so you can log Identity Manager's SPML traffic and to help you diagnose problems.

Method 1

The `SpmlClient` and `LighthouseClient` classes both provide a `setTrace` method that takes a Boolean argument. If you enable this `setTrace` method, the XML for the request sent by the client and the XML for the response received from the server will be printed to the *client* console as they are sent and received.
For example

```
SpmlClient client = new SpmlClient();
client.setURL("http://www.company.com/idm/spml");
client.setTrace(true);
```

Method 2

To turn tracing on for an individual SPML RPC request, you can pass the `trace` operational attribute to the RPC request on the server side.

Tracing occurs during servlet initialization, and it controls how information is output for the RPC traffic of a servlet handling SPMLv2 requests. For example, the trace prints the raw XML that is sent back and forth on whatever the `System.out` is for that servlet (which is a function of the App container). For example:

```
AddRequest ar = new AddRequest();
ar.setOperationalAttribute("trace", "true");
```

How using the `trace` attribute will affect server operation is vendor-specific. Currently, Identity Manager prints the raw request and response data to the *server* console, which is useful if the client application is not associated with a console window.

The code is not yet implemented in Identity Manager, so for more information consult your OpenSPML Toolkit product documentation.

Examples

This section provides the following examples to illustrate several common methods for implementing SPML:

- Add Request
- Modify Request
- Search Request

Add Request

An example Add Request is shown in Code Example 7-16:

Code Example 7-16 Add Request

```
SpmlClient client = new SpmlClient();
   client.setURL("http://www.company.com/idm/spml");

   AddRequest req = new AddRequest();

   req.setObjectClass("person");
   req.setIdentifier("maurelius");
   req.setAttribute("gn", "Marcus");
   req.setAttribute("sn", "Aurelius");
   req.setAttribute("email", "maurelius@example.com");

   SpmlResponse res = client.request(req);

   if (res.getResult() .equals(SpmlResponse.RESULT_SUCCESS))
      System.out.println("Person was successfully created");
```

Modify Request

This section provides two example Authenticated SPML Modify Requests.

The only difference between these examples is that the Code Example 7-18 uses the `LighthouseClient` class and two additional method calls to `client.setUser` and `client.setPassword`.

Code Example 7-17 Authenticated SPML Request

```
SpmlClient client = new SpmlClient();
   client.setURL("http://www.company.com/idm/spml");
   ModifyRequest req = new ModifyRequest();
   req.setIdentifier("maurelius");
   req.setModification("email", "marcus.aurelius@example.com");
   SpmlResponse res = client.request(req);
   if (res.getResult() .equals(SpmlResponse.RESULT_SUCCESS))
      System.out.println("Person was successfully modified");
```

Code Example 7-18 Authenticated SPML Request with LighthouseClient

```
LighthouseClient client = new LighthouseClient();
   client.setURL("http://www.company.com/idm/spml");
   client.setUser("maurelius");
   client.setPassword("xyzzy");
   ModifyRequest req = new ModifyRequest();
   req.setIdentifier("maurelius");
   req.setModification("email", "marcus.aurelius@example.com");
   SpmlResponse res = client.request(req);
   if (res.getResult() .equals(SpmlResponse.RESULT_SUCCESS))
      System.out.println("Person was successfully modified");
```

Search Request

An example Search Request is shown in Code Example 7-19:

Code Example 7-19 Search Request

```
SpmlClient client = new SpmlClient();
   client.setURL("http://www.company.com/idm/spml");
   SearchRequst req = new SearchRequest();
   // specify the attributes to return
   req.addAttribute("sn");
   req.addAttribute("email");
   // specify the filter
   FilterTerm ft = new FilterTerm();
   ft.setOperation(FilterTerm.OP_EQUAL);
   ft.setName("gn");
   ft.setValue("Jeff");
   req.addFilter(ft);
```

Code Example 7-19 Search Request *(Continued)*

```
SearchResponse res = (SearchResponse)client.request(req);
// display the results
List results = res.getResults();
if (results != null)  {
   for (int i = 0 ; i < results.size() ; i++) {
      SearchResult sr = (SearchResult)results.get(i);
      System.out.println("Identifier=" +
                         sr.getIdentifierString() +
                         " sn=" +
                         sr.getAttribute("sn") +
                         " email=" +
                         sr.getAttribute("email"));
      }
}
```

Chapter 8

Using SPML 2.0 with Identity Manager Web Services

This chapter describes SPML 2.0 support in Identity Manager 7.1; including which features are supported and why, how to configure SPML 2.0 support, and how to extend support in the field.

NOTE This chapter focuses exclusively on SPML 2.0. Unless noted otherwise, all references to SPML in this chapter indicate the *2.0* version.

You should also read Chapter 7, "Using SPML 1.0 with Identity Manager Web Services," which also contains useful information about using SPML.

This information is organized as follows:

Who Should Read This Chapter

Application developers and developers who are responsible for integrating Identity Manager can use the SPML classes described in this chapter to format service provisioning request messages and to parse response messages.

Overview

Identity Manager's Web services are SOAP-based programmatic interfaces that enable Identity Manager to communicate with other Web-based applications — without being restricted to a particular operating system or programming language.

Web services interfaces are hosted on a Web server, and their functions are exposed through the following open standards over an Internet protocol framework:

- **XML**: Used to tag data
- **SOAP**: Used to encode and transfer data over the Internet
- **WSDL**: Used to describe available services
- **UDDI**: Used to list which services are available

You can access Identity Manager Web services using SOAP messages over an HTTP connection.

How SPML 2.0 Compares to SPML 1.0

Identity Manager Web services support both SPML version 1.0 and version 2.0 protocols (open standards for service provisioning using XML) for communication with provisioning systems.

NOTE See Chapter 7, "Using SPML 1.0 with Identity Manager Web Services" for information about using SPML version 1.0 with Identity Manager.

SPML 2.0 offers many improvements over SPML 1.0, including:

- Where SPML 1.0 has been called a slightly improved DSML, SPML 2.0 defines an extensible protocol (through *Capabilities*) with support for a DSML profile, as well as *XML Schema* profiles. SPML 2.0 differentiates between the protocol and the data it carries.
- An SPML 2.0 provider can provide multiple *targets*, or endpoints.
- The SPML 2.0 protocol enables better interoperability between vendors — especially for the Core capabilities (those found in 1.0).

  You can "extend" SPML 1.0 using `ExtendedRequest`, but there is no guidance about what those requests can be. SPML 2.0 defines a set of "standard capabilities" that allow you to add support in well-defined ways.
- SPML 2.0 provides additional capabilities that enable progressive support (see Table 8-1)

Table 8-1 SPML Capabilities

SPML 1.0	SPML 2.0
Add	Add
Modify	Modify
Delete	Delete
Lookup	Lookup
SchemaRequest	ListTargets
Search	Search as a "standard" Capability (*not supported this release)*
ExtendedRequest	Captured in "standard" Capabilities: • Async: Asynchronous processing of requirements • Batch: Process a batch of requests • BulK: Process modifies or deletes using iteration • Password: Change, set, reset, validate, or expire passwords • Reference: Refer to PSOs between targets • Search: Search for PSOs • Suspend: Enable or disable PSOs • Update: Find change records for objects that have been updated (can also be captured in "custom" Capabilities.)

How SPML 2.0 Concepts Are Mapped to Identity Manager

SPML 2.0 uses its own terminology to discuss the objects that are managed by a provisioning system — defining terms like *PSO* and *psoID*. This section discusses the how these concepts are mapped into Identity Manager.

Target

A *target* is a logical end-point in the server. Each target is named and declares the schema of the objects (see the following "PSO" section) that it manages. The target also declares which capabilities (set of requests) are supported.

Currently, Identity Manager supports only *one* target — you cannot declare multiple targets. You can name this target anything you want, but the data objects' format must conform to the DSML-profile.

The supported target is the one target defined in the `spml2.xml` file (`Configuration:SPML2` object). For example, in Code Example 8-6 on page 278 `ListTargetResponse` returns one target, `spml2-DSML-Target`.

PSO

As mentioned in the previous section, targets manage *PSOs*. A PSO (Provisioning Service Object) is somewhat analogous to a *view* in Identity Manager, but without behavior. Consequently, you can think of a PSO as the data portion of an Identity Manager view; a user view in particular.

NOTE As of this release, Identity Manager only manages Users and requires you to define a `UserExtendedAttribute` called *`objectclass`*.

For Identity Manager's purposes, a PSO is a collection of attributes that are mapped (via a form) to and from a user view. Each object specifies an `objectclass` attribute that is used to map the object to an `objectclass` definition in the schema defined for the target. This attribute is used in turn to find a `repoType` and a form that maps the attributes to and from the Identity Manager view.

PSOIdentifier

SPML includes an object ID, called a *PsoID*.

Because the specification recommends that a PSOIdentifier (PsoID) be opaque to a requestor (client), Identity Manager uses its repository ID (repoID) as the PsoID when adding PSOs to the system.

The repoID is distinct and is not meant for presentation to a user. When a requestor displays a PSO to a user, it should use the equivalent of the `waveset.accountid` (or whatever attributes are used in the Identity template) to present the object's ID.

When identifying the PSO (as in a `ModifyRequest`), the requestor should use the repoID and not the `waveset.accountId`. Although the requestor can use the `waveset.accountId` as a PSOIdentifier, doing so is not recommended and it may change in a future release. Requestors should try to keep the PsoID opaque.

PSOs use an `objectclass` attribute to specify the object type. For Identity Manager this attribute must be a `UserExtendedAttribute`. If this attribute is not there when a request is made, Identity Manager allows you to specify a "default" to use. You might not see an `objectclass` attribute for users that existed before you enabled SPML 2.0.

Open Content and Operational Attributes

SPML makes heavy use of `xsd:any` in the `.xsds` to provide what the specification refers to as *Open Content*. In SPML, Open Content means that most elements can contain elements of any type. Identity Manager uses this idea to provide *OperationalNVPs* (`NameValuePairs`) and *OperationalAttributes* that control processing. OperationalNVPs appear as elements in the XML, while Operational Attributes appear as attributes. See the OpenSPML 2.0 Toolkit at `http://www.openspml.org` for more information.

OperationalNVPs and Operational Attributes are discussed further in the"Supported SPML 2.0 Capabilities," section; however, you use one NVP in all requests (except `ListTargets`) and in *all* responses. Identity Manager stores a `sessionToken` in an OperationalNVP called `session`, which allows the system to cache sessions on behalf of the user and improves efficiency.

Supported SPML 2.0 Capabilities

Identity Manager 7.0 supports all of the Core capabilities in the SPML 2.0 specification, using the DSML profile. Identity Manager also supports some of the optional Standard capabilities (such as Batch and Async) and partially supports some Standard capabilities (such as Bulk).

This section describes which SPML 2.0 capabilities are supported in Identity Manager 7.0, where Identity Manager (knowingly) varies from the specification and profile document, and which operational attributes are required by Identity Manager.

Supported Core Capabilities

Identity Manager supports the following Core capabilities:

Table 8-2 Core Capabilities

Capability	Description	Operational Attributes	Caveats
AddRequest	Adds a specified PSO to the system.	None	Identity Manager officially supports only a single target.
DeleteRequest	Deletes a specified PSO from the system.	None	Identity Manager officially supports only a single target.
ListTargetsRequest	Lists the targets that are available through Identity Manager.	• `username`: Specifies the name of the user. • `password`: Specifies the password to use for establishing a session.	• Identity Manager officially supports a single target. • Identity Manager does not require you to use `listTargets` as the first call in a conversation; however, it does allow `operationalAttributes` on this request to specify a username/password pair for establishing a session with the server. (You can also use `Waveset.properties`.) In general, it is more efficient to login and use the session token. Identity Manager provides a class called `SessionAwareSpml2Client` for this purpose.
LookupRequest	Finds and returns the attributes of the named PSO.	None	None
ModifyRequest	Modifies specified PSO attributes.	None	Due to a discrepancy between the main SPML 2.0 specification and the DSML Profile specification, Identity Manager *does not* support `select` (and `component`, etc.). Instead Identity Manager uses the DSML Modification Mode and elements according to the DSML Profile.

NOTE General caveats include:

- Identity Manager supports only the DSML Profile.
- Identity Manager does not require `listTargets` to be the first call in a conversation; however, it does allow `operationalAttributes` on this request to specify a `username/password` pair for establishing a session with the server. (You can also use `Waveset.properties`.)

`AddRequest` and `ListTargetRequest` examples follow.

AddRequest Examples

This section provides several AddRequest examples.

Code Example 8-1 is a `.jsp` that invokes a `ListTargetsRequest` through Identity Manager's `SessionAwareSpml2Client` class.

Code Example 8-1 Example Client Code

```
<%@page contentType="text/html"%>
<%@page import="org.openspml.v2.client.*,
                com.sun.idm.rpc.spml2.SessionAwareSpml2Client"%>
<%@page import="org.openspml.v2.profiles.dsml.*"%>
<%@page import="org.openspml.v2.profiles.*"%>
<%@page import="org.openspml.v2.util.xml.*"%>
<%@page import="org.openspml.v2.msg.*"%>
<%@page import="org.openspml.v2.msg.spml.*"%>
<%@page import="org.openspml.v2.util.*"%>

<%
final String url = "http://localhost:8080/idm/servlet/openspml2";
%>

<html>
<head><title>SPML2 Test</title></head>
<body>
<%

 // need a client.
 SessionAwareSpml2Client client = new SessionAwareSpml2Client( url );

 // login
 client.login("configurator", "password");

 // AddRequest
 String rid = "rid-spmlv2"; // The RequestId is not strictly required.
```

Code Example 8-1 Example Client Code*(Continued)*

```
 Extensible data = new Extensible();
 data.addOpenContentElement(new DSMLAttr("accountId", user));
 data.addOpenContentElement(new DSMLAttr("objectclass", "spml2Person"));
 data.addOpenContentElement(new DSMLAttr("credentials", password));

 AddRequest add = new AddRequest(rid, // String requestId,
                ExecutionMode.SYNCHRONOUS, // ExecutionMode executionMode,
                null, // PSOIdentifier type,
                null, // PSOIdentifier containerID,
                data, // Extensible data,
                null, // CapabilityData[] capabilityData,
                null,  // String targetId,
                null  // ReturnData returnData
            );

    // Submit the request
    Response res = client.send( add );
%>
<%= res.toString()%>
</body>
</html>
```

Code Example 8-2 shows the body of the SOAP message that was sent.

Code Example 8-2 Example Request XML

```
<addRequest xmlns='urn:oasis:names:tc:SPML:2:0' requestID='rid-spmlv2'
    executionMode='synchronous'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
     name='session' value='AAALPgAAYD0A...'/>
  <data>
    <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='accountId'>
      <dsml:value>exampleSpml2Person</dsml:value>
    </dsml:attr>
    <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='objectclass'>
      <dsml:value>spml2Person</dsml:value>
    </dsml:attr>
    <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='credentials'>
      <dsml:value>pwdpwd</dsml:value>
    </dsml:attr>
  </data>
</addRequest>
```

Code Example 8-3 shows the body of the SOAP message that is returned to the client.

Code Example 8-3 Example Response XML

```
<addResponse xmlns='urn:oasis:names:tc:SPML:2:0' status='success' requestID='rid-spmlv2'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
      name='session' value='AAALPgAAYD0A...'/>
  <pso>
    <psoID ID='anSpml2Person'/>
    <data>
      <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='accountId'>
        <dsml:value>anSpml2Person</dsml:value>
      </dsml:attr>
      <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='objectclass'>
        <dsml:value>spml2Person</dsml:value>
      </dsml:attr>
      <dsml:attr xmlns:dsml='urn:oasis:names:tc:DSML:2:0:core' name='credentials'>
        <dsml:value>pwdpwd</dsml:value>
      </dsml:attr>
    </data>
  </pso>
</addResponse>
```

ListTargetsRequest Examples

The following examples show ListsTargetRequest that are available via Identity Manager.

Code Example 8-4 shows a `.jsp` invokes a `ListTargetsRequest` via Identity Manager's `SessionAwareSpml2Client` class.

Code Example 8-4 Example Client Code

```
<%@page contentType="text/html"%>
<%@page import="org.openspml.v2.client.*,
                com.sun.idm.rpc.spml2.SessionAwareSpml2Client"%>
<%@page import="org.openspml.v2.profiles.dsml.*"%>
<%@page import="org.openspml.v2.profiles.*"%>
<%@page import="org.openspml.v2.util.xml.*"%>
<%@page import="org.openspml.v2.msg.*"%>
<%@page import="org.openspml.v2.msg.spml.*"%>
<%@page import="org.openspml.v2.util.*"%>

<%
final String url = "http://localhost:8080/idm/servlet/openspml2";
%>
```

Code Example 8-4 Example Client Code*(Continued)*

```
<html>
<head><title>SPML2 Test</title></head>
<body>
<%

 // need a client.
 SessionAwareSpml2Client client = new SessionAwareSpml2Client( url );

 // login (sends a ListTargetsRequest)
 Response res = client.login("configurator", "password");

%>
<%= res.toString()%>
</body>
</html>
```

Code Example 8-5 shows the body of the SOAP message that is sent.

Code Example 8-5 Example Request XML

```
<listTargetsRequest xmlns='urn:oasis:names:tc:SPML:2:0' requestID='rid[7013]'
    executionMode='synchronous'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
    name='accountId' value='configurator'/>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
    name='password' value='password'/>
</listTargetsRequest>
```

Code Example 8-6 shows the body of the SOAP message that is received by or returned to the client.

Code Example 8-6 Example Response XML

```
<listTargetsResponse xmlns='urn:oasis:names:tc:SPML:2:0' status='success' requestID='rid[6843]'>
  <openspml:operationalNameValuePair xmlns:openspml='urn:org:openspml:v2:util:xml'
     name='session' value='AAALPgAAYD0A...'/>
  <target targetID='spml2-DSML-Target' profile='urn:oasis:names:tc:SPML:2:0:DSML'>
    <schema>
      <spmldsml:schema xmlns:spmldsml='urn:oasis:names:tc:SPML:2:0:DSML'>
        <spmldsml:objectClassDefinition name='spml2Person'>
          <spmldsml:memberAttributes>
            <spmldsml:attributeDefinitionReference required='true' name='objectclass'/>
```

Code Example 8-6 Example Response XML*(Continued)*

```
            <spmldsml:attributeDefinitionReference required='true' name='accountId'/>
            <spmldsml:attributeDefinitionReference required='true' name='credentials'/>
            <spmldsml:attributeDefinitionReference name='firstname'/>
            <spmldsml:attributeDefinitionReference name='lastname'/>
            <spmldsml:attributeDefinitionReference name='emailAddress'/>
          </spmldsml:memberAttributes>
        </spmldsml:objectClassDefinition>
        <spmldsml:attributeDefinition name='objectclass'/>
        <spmldsml:attributeDefinition description='Account Id' name='accountId'/>
        <spmldsml:attributeDefinition description='Credentials, e.g. password'
            name='credentials'/>
        <spmldsml:attributeDefinition description='First Name' name='firstname'/>
        <spmldsml:attributeDefinition description='Last Name' name='lastname'/>
        <spmldsml:attributeDefinition description='Email Address' name='emailAddress'/>
      </spmldsml:schema>
      <supportedSchemaEntity entityName='spml2Person'/>
    </schema>
    <capabilities>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:async'/>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:batch'/>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:bulk'/>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:pass'/>
      <capability namespaceURI='urn:oasis:names:tc:SPML:2:0:suspend'/>
    </capabilities>
  </target>
</listTargetsResponse>
```

Async Capability Support

Identity Manager supports the Async capabilities described in Table 8-3:

Table 8-3 Async Capabilities

Capability	Description	Operational Attributes	Caveats
CancelRequest	Cancels a request, using the request ID.	None	
StatusRequest	Returns the status of a request, using the request ID.	None	

Batch Capability Support

Identity Manager supports the Batch capability described in Table 8-4.

Table 8-4 Batch Capability

Capability	Description	Operational Attributes	Caveats
BatchRequest	Executes a batch of requests.	None	

Bulk Capability Support

Identity Manager supports the Bulk capabilities described in Table 8-5:

Table 8-5 Bulk Capabilities

Capability	Description	Operational Attributes	Caveats
BulkDeleteRequest	Executes a bulk delete of PSOs.	None	
BulkModifyRequest	Executes a bulk modify of matching PSOs.	None	

Password Capability Support

Identity Manager supports the Password capabilities described in Table 8-6:

Table 8-6 Password Capabilities

Capability	Description	Operational Attributes	Caveats
ExpirePasswordRequest	Expires a password.	None	• You cannot specify resources/targets. Doing so causes the Identity Manager User object password to expire; which then causes the password on all user's resources to expire. • Identity Manager does not support the `remainingLogins` attribute. If you set this attribute to anything other than the default, an `OperationNotSupported` error occurs.

Table 8-6 Password Capabilities *(Continued)*

Capability	Description	Operational Attributes	Caveats
ResetPasswordRequest	Resets the password and returns the new value on all accounts.	None	Passwords are sensitive. Use SSL or some other secure transport.
SetPasswordRequest	Sets the password.	None	Passwords are sensitive. Use SSL or some other secure transport.
ValidatePasswordRequest	Determines whether the given password is valid.	None	Passwords are sensitive. Use SSL or some other secure transport.

Example Password capabilities follow.

ResetPasswordRequest Example

Code Example 8-7 is an example ResetPasswordRequest.

Code Example 8-7 Example ResetPasswordRequest

```
ResetPasswordRequest rpr = new ResetPasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
rpr.setPsoID(psoId);
...
```

SetPasswordRequest Example

Code Example 8-8 is an example SetPasswordRequest.

Code Example 8-8 Example SetPasswordRequest

```
SetPasswordRequest spr = new SetPasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
spr.setPsoID(psoId);
spr.setPassword("newpassword");
spr.setCurrentPassword("oldpassword");
...
```

ValidatePasswordRequest Example

Code Example 8-9 is an example ValidatePasswordRequest.

Code Example 8-9 Example ValidatePasswordRequest

```
ValidatePasswordRequest vpr = new ValidatePasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
vpr.setPsoID(psoId);
vpr.setPassword("apassword");
...
```

Suspend Capability Support

Identity Manager supports the Suspend capabilities described in Table 8-7.

Table 8-7 Suspend Capabilities

Capability	Description	Operational Attributes	Caveats
ResumeRequest	Resumes (enables) a PSO User.	None	Does not support `EffectiveDate`. If you set `EffectiveDate`, Identity Manager returns an `OperationNotSupported` error.
SuspendRequest	Suspends an accounts/PSO (disable)	None	Does not support `EffectiveDate`. If you set `EffectiveDate`, Identity Manager returns an `OperationNotSupported` error.

Which SPML 2.0 Features Are Not Supported

Identity Manager 7.1 does not support the Reference capability, the Search capability, or the Updates capability.

In addition, none of the supported capabilities use the `CapabilityData` class, so there is no infrastructure in the Identity Manager code base to support it, which is significant if you want to implement custom capabilities. Note that the OpenSPML 2.0 Toolkit does support `CapabilityData` in the marshallers, unmarshallers, and so forth.

Why No References?

Identity Manager does not support references for two reasons; the Reference Capability is typically used to refer across targets or to refer between objects on the same target.

Because Identity Manager officially only supports one target, the first use-case is out. Additionally, because User is the only `repoType` supported by Identity Manager (mapped to a PSO type using `objectclass`), and these objects do not refer to other objects, there is no need to support References at this time.

Why No Search Capability?

Currently, Search support is provided only through SPML 1.0.

Identity Manager does not support the SPML 2.0 Search Capability because the capability does not perform a search very efficiently yet. In short, the Search filters operate on the full User View and therefore need to instantiate every User object into a full view; which is not very efficient, but it is necessary as a provisioning system. Because of this, Identity Manager cannot reasonably provide iteration (or an efficient search) over the results of the search. Adding the Search capability will be addressed in a future release.

Why No Updates?

Identity Manager does not support the Updates Capability because it does not track Change Records, which are required information for this capability.

Configuring Identity Manager to Use SPML 2.0

When configuring an Identity Manager server to use SPML 2.0, the first step is to decide which attributes you want to manage through your target.

NOTE You can have more than one attribute in the target.

Decide which attribute sets (`objectclasses`) the interface clients will use when managing users in the Identity Manager instance that uses this interface. This set of attributes is a *PSO*.

You must also know how to map those attributes to and from a User View using a form.

This section describes how to configure a system that uses PSOs containing the following attributes for a DSML objectclass called `spml2Person`:

- `accountId`
- `objectclass`
- `credentials`
- `firstname`
- `lastname`
- `emailAddress`

These attributes are mapped to the User View. This section also provides short examples to demonstrate how to manage these PSOs using SPML 2.0 support in Identity Manager. Note that this configuration is delivered in the `sample/spml2.xml` file provided with the product. You can see this file for detailed information.

After deciding on the format of a PSO; enable the service as described in the next section, which discusses `web.xml` and what has been added for SPML 2.0.

The SPML2 Configuration Object

The out-of-the-box configuration for SPML 2.0 support is provided in `sample/spml2.xml`. This file defines the objects that Identity Manager needs to support SPML 2.0.

One of these Configuration type objects, named `SPML2`, is discussed in this section. You use this object to change how SPML 2.0 support behaves or to extend the system. (A separate section will discuss extensions in more detail.)

The following example is an annotated export of the object.

Code Example 8-10 Annotated Export of the SPML2.XML Object

```
<Configuration name='SPML2' authType='SPML'>
    <Extension>
        <Object>

            <!-- Each of the objects in this list represents a tuple of an "executorClass"
                 and the set of Requests that it knows how to handle. A Request type can
                 appear in more than one executor tuple.  This results in a chain of
                 executors for that type.  If the first one cannot handle a given instance
                 of that type, the request is passed to the next one in the chain.
                 It's important to note that this list (executors) is processed in order.
                 Generally, you'll leave this section alone. -->

            <Attribute name='executors'>
                <List>
                    <Object name='com.sun.idm.rpc.spml2.core.ListTargetsExecutor'>
                        <Attribute name="requests" value="org.openspml.v2.msg.
                             spml.ListTargetsRequest"/>
                    </Object>
                    <Object name='com.sun.idm.rpc.spml2.pass.PasswordRequestExecutor'>
                        <Attribute name='requests'>
                            <List>
                               <String>org.openspml.v2.msg.pass.SetPasswordRequest</String>
                             <String>org.openspml.v2.msg.pass.ResetPasswordRequest</String>
 <String>org.openspml.v2.msg.pass.ValidatePasswordRequest</String>
 <String>org.openspml.v2.msg.pass.ExpirePasswordRequest</String>
                            </List>
                        </Attribute>
                    </Object>
                    <Object name='com.sun.idm.rpc.spml2.core.AddRequestExecutor'>
                        <Attribute name="requests"
                             value='org.openspml.v2.msg.spml.AddRequest'/>
                    </Object>
                    <Object name='com.sun.idm.rpc.spml2.core.DeleteRequestExecutor'>
                        <Attribute name='requests'
                             value='org.openspml.v2.msg.spml.DeleteRequest'/>
                    </Object>
```

Code Example 8-10 Annotated Export of the SPML2.XML Object*(Continued)*

```
                <Object name='com.sun.idm.rpc.spml2.core.LookupRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spml.LookupRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.core.ModifyRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spml.ModifyRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.suspend.SuspendRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlsuspend.SuspendRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.suspend.ResumeRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlsuspend.ResumeRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.suspend.ActiveRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlsuspend.ActiveRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.batch.BatchRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlbatch.BatchRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.bulk.BulkDeleteRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlbulk.BulkDeleteRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.bulk.BulkModifyRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlbulk.BulkModifyRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.async.StatusRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlasync.StatusRequest'/>
                </Object>
                <Object name='com.sun.idm.rpc.spml2.async.CancelRequestExecutor'>
                    <Attribute name='requests'
                         value='org.openspml.v2.msg.spmlasync.CancelRequest'/>
                </Object>
            </List>
        </Attribute>
        <Attribute name='targets'>
            <List>

                <!-- This is the name of the target; broken out so it easy to read
                     and refer too. This becomes the targetID as the name of this Object
                     replaces the $OBJECT_NAME$ variable in 'xmlTemplate'. -->

                <Object name="spml2-DSML-Target">
```

Code Example 8-10 Annotated Export of the SPML2.XML Object*(Continued)*

```
                        <!-- You can state that the Target is attached to a profile;
                             replaces $PROFILE_ATTRIBUTE$ with 'profile="<value>"' in the
                             'xmlTemplate'.
                        -->

                       <Attribute name="profile" value="urn:oasis:names:tc:SPML:2:0:DSML"/>

                      <!-- This is the target definition; the first two lines, and the last,
                           are always the same (for each Object in the "targets" list) -->

                        <Attribute name='xmlTemplate'>
                            <String><![CDATA[
<target targetID="$OBJECT_NAME$" $PROFILE_ATTRIBUTE$>
    <schema>
        <spmldsml:schema xmlns:spmldsml="urn:oasis:names:tc:SPML:2:0:DSML">
            <spmldsml:attributeDefinition name="objectclass"/>
            <spmldsml:attributeDefinition name="accountId" description="Account Id"/>
            <spmldsml:attributeDefinition name="credentials" description="Credentials,
                e.g. password"/>
            <spmldsml:attributeDefinition name="firstname" description="First Name"/>
            <spmldsml:attributeDefinition name="lastname" description="Last Name"/>
            <spmldsml:attributeDefinition name="emailAddress" description="Email Address"/>
            <spmldsml:objectClassDefinition name="spml2Person">
                <spmldsml:memberAttributes>
                    <spmldsml:attributeDefinitionReference name="objectclass"
                        required="true"/>
                  <spmldsml:attributeDefinitionReference name="accountId" required="true"/>
                    <spmldsml:attributeDefinitionReference name="credentials"
                         required="true"/>
                    <spmldsml:attributeDefinitionReference name="firstname"/>
                    <spmldsml:attributeDefinitionReference name="lastname"/>
                    <spmldsml:attributeDefinitionReference name="emailAddress"/>
                </spmldsml:memberAttributes>
            </spmldsml:objectClassDefinition>
        </spmldsml:schema>
        <supportedSchemaEntity entityName="spml2Person"/>
    </schema>
    <capabilities>
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:async"/>
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:batch"/>
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:bulk"/>
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:pass"/>
        <capability namespaceURI="urn:oasis:names:tc:SPML:2:0:suspend"/>
    </capabilities>
</target>
                            ]]></String>
                        </Attribute>
                    </Object>
                </List>
            </Attribute>
```

Code Example 8-10 Annotated Export of the SPML2.XML Object*(Continued)*

```
                    <!-- This is like the "classes" list in the spml.xml config. We added the
                         ability to apply a mapping to one or more targets, in case the they
                         have common objectclass / type names.  See the SPML 1.0 file and
                         docs for more info.
            -->

            <Attribute name='mappings'>
                <List>
                    <Object name='spml2Person'>
                        <Attribute name='type' value='User'/>
                        <Attribute name='form' value='spml2PersonForm'/>
                        <Attribute name='default' value='true'/>
                        <Attribute name='targets'>
                            <String>"spml2-DSML-Target"</String>
                        </Attribute>
                    </Object>
                    <Object name='request'>
                        <Attribute name='type' value='TaskInstance'/>
                        <Attribute name='filter'>
                            <AttributeCondition attrName='defName' operator='equals'
                                operand='SPML2Request'/>
                        </Attribute>
                    </Object>
                </List>
            </Attribute>
        </Object>
    </Extension>
</Configuration>

<!-- We ALSO need to define the Form that maps the attributes in our objectclass(es) to and
from the view.  The following form does this for the spml2Person objectclass.-->

<Configuration name='spml2PersonForm' authType='SPML'>
    <Extension>
        <Form>
            <Field name='accountId'>
                <Derivation>
                    <ref>waveset.accountId</ref>
                </Derivation>
            </Field>
            <Field name='waveset.accountId'>
                <Expansion>
                    <ref>accountId</ref>
                </Expansion>
            </Field>
            <Field name='emailAddress'>
                <Derivation>
                    <ref>waveset.email</ref>
                </Derivation>
            </Field>
            <Field name='global.email'>
                <Expansion>
                    <ref>emailAddress</ref>
```

Code Example 8-10 Annotated Export of the SPML2.XML Object*(Continued)*

```
                </Expansion>
            </Field>
            <Field name='objectclass'>
                <Derivation>
                    <ref>accounts[Lighthouse].objectclass</ref>
                </Derivation>
            </Field>
            <Field name='accounts[Lighthouse].objectclass'>
                <Expansion>
                    <ref>objectclass</ref>
                </Expansion>
            </Field>
            <Field name='credentials'>
                <Derivation>
                    <ref>password.password</ref>
                </Derivation>
            </Field>
            <Field name='password.password'>
                <Expansion>
                    <ref>credentials</ref>
                </Expansion>
            </Field>
            <Field name='lastname'>
                <Derivation>
                    <ref>accounts[Lighthouse].lastname</ref>
                </Derivation>
            </Field>
            <Field name='global.lastname'>
                <Expansion>
                    <ref>lastname</ref>
                </Expansion>
            </Field>
            <Field name='firstname'>
                <Derivation>
                    <ref>accounts[Lighthouse].firstname</ref>
                </Derivation>
            </Field>
            <Field name='global.firstname'>
                <Expansion>
                    <ref>firstname</ref>
                </Expansion>
            </Field>
        </Form>
    </Extension>
</Configuration>
```

web.xml

The following section in `web.xml` sets up the `openspmlRouter` servlet, which is the servlet that handles SPML 2.0 requests.

NOTE This section of `web.xml` ships with a default installation, and no action is required for this component.

Code Example 8-11 Setting-Up the openspmlRouter Servlet

```
<servlet>
    <servlet-name>openspmlRouter</servlet-name>
    <display-name>OpenSPML SOAP Router</display-name>
    <description>A router of RPC traffic - nominally SPML 2.0 over SOAP</description>
    <servlet-class>
        org.openspml.v2.transport.RPCRouterServlet
    </servlet-class>
    <init-param>
        <param-name>dispatchers</param-name>
        <param-value>org.openspml.v2.transport.SPMLViaSoapDispatcher</param-value>
    </init-param>
    <init-param>
        <param-name>trace</param-name>
        <param-value>false</param-value>
    </init-param>
    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberMarshaller</param-value>
    </init-param>
    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers.UberMarshaller.trace</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>SpmlViaSoap.spmlExecutors</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberExecutor</param-value>
    </init-param>
</servlet>
```

This file contains an optional `init-param` that you can add to open a monitor window (in Swing) that displays the flow of SPML 2.0 messages. You will find this monitor window useful for debugging.

The following example shows you what to add.

```
<init-param>
    <param-name>monitor</param-name>
 <param-value>org.openspml.v2.util.SwingRPCRouterMonitor</param-value>
</init-param>
```

In Code Example 8-12, the section is commented and provides information about the other `init-params`.

Code Example 8-12 Commented Example

```
<servlet>
    <servlet-name>openspmlRouter</servlet-name>
    <display-name>OpenSPML SOAP Router</display-name>
    <description>A router of RPC traffic - nominally SPML 2.0 over SOAP</description>
    <servlet-class>
    org.openspml.v2.transport.RPCRouterServlet
    </servlet-class>

    <!--
        The Router uses dispatchers to process SOAP messages. This is one that is in the
        toolkit that knows about SOAP. It has its own parameters, via naming convention.
        See below.
    -->

    <init-param>
        <param-name>dispatchers</param-name>
        <param-value>org.openspml.v2.transport.SPMLViaSoapDispatcher</param-value>
    </init-param>

    <!--
        Turn on trace to have the servlet write informational messages to the log.
    -->

    <init-param>
        <param-name>trace</param-name>
        <param-value>false</param-value>
    </init-param>

    <!--
          The SpmlViaSOAPDispatcher (yes, the one above) uses marshallers; there can be
          a chain, to move XML to SPML objects and back. We use one; we implemented
          UberMarshaller for this purpose.  It's really a composition of toolkit classes.
    -->
    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberMarshaller</param-value>
```

Code Example 8-12 Commented Example*(Continued)*

```
    </init-param>

    <!--
          Our marshaller (UberMarshaller) has its own trace setting; which doesn't really
          do anything in this release
    -->

    <init-param>
        <param-name>SpmlViaSoap.spmlMarshallers.UberMarshaller.trace</param-name>
        <param-value>true</param-value>
    </init-param>

    <!--
      Finally, the dispatcher has a list of executors that actually implement the
      functionality.  So, it sees a request, takes the SOAP envelope off, take the body
      from XML to OpenSPML Request classes, and then asks the list of executors if they can
      process it. We provided one, UberExecutor.  It will redispatch the request to our
      other executors.  Those are specified in spml2.xml (Configuration:SPML2).
    -->

    <init-param>
        <param-name>SpmlViaSoap.spmlExecutors</param-name>
        <param-value>com.sun.idm.rpc.spml2.UberExecutor</param-value>
    </init-param>
</servlet>
```

Using Trace with SPML

SPML provides the following options for turning on trace output so you can log Identity Manager's SPML traffic and diagnose problems.

Method 1

To turn tracing on for an individual SPML RPC request, you can pass the `trace` operational attribute to the RPC request on the server side.

```
AddRequest ar = new AddRequest();
ar.setOperationalAttribute("trace", "true");
```

Tracing controls what information is output for the RPC traffic of a servlet handling SPMLv2 requests. For example:

How using the `trace` attribute will affect server operation is vendor-specific. Currently, Identity Manager prints the raw XML request and response data to the *server* console, which is useful if the client application is not associated with a console window.

For more information consult your OpenSPML toolkit product documentation.

Method 2

You can also turn tracing on by initializing the SOAP `rpcrouter` servlet, which controls the output of RPC traffic information for the servlet handling SPML requests. The `rpcrouter` servlet takes an <*init parameter*> that enables SOAP tracing on the server side. The servlet's initialization logic checks for a `trace` configuration parameter and then, if the parameter's value is `true`, prints the raw request and response data to the console.

For example, it prints the raw XML that is sent back and forth to whatever `System.out` has been specified for that servelet. (Where `System.out` is a function of the application container.)

NOTE The SOAP `rpcrouter` servlet is a third-party, open source `org.openspml.server.SOAPRouter` class from the SPML organization. For more information about this servlet, refer to your OpenSPML Toolkit documentation.

The next section describes `spml2.xml`, with some annotations.

Extending the System

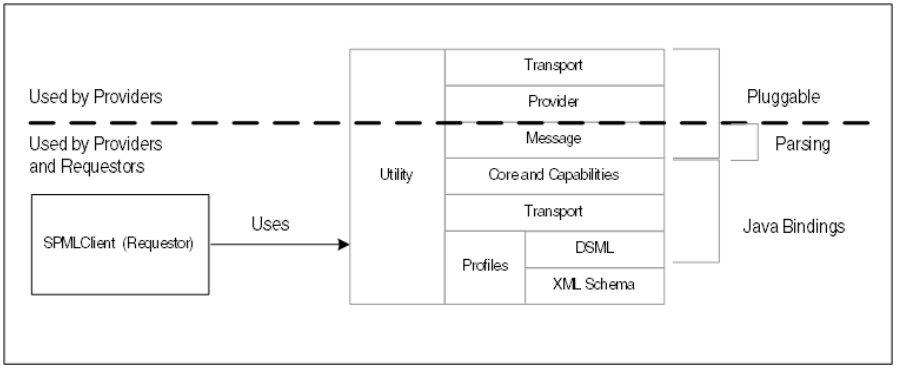
You extend the schema by modifying the configuration object, and you can add executors for requests by changing the section. Using forms, you can map DSML to Views and back.

It is less obvious, but you can also replace the dispatcher, marshaller, and the UberExecutor, with those of your own devising.

- If you do not want to use SOAP, just replace the dispatcher in the first case.
- If you do not want to use HTTP, replace the router with a different kind of servlet.
- If you want different XML parsing, replace the Marshaller with your own.

SPML 2.0 provides a wide-open array of pluggability, which is due to Identity Manager's use of the OpenSPML 2.0 Toolkit. Figure 8-1 illustrates the OpenSPML 2.0 Toolkit architecture.

Figure 8-1 OpenSPML 2.0 Toolkit Architecture



Sample SPML 2.0 Adapter

Identity Manager provides a sample SPML 2.0 resource adapter that can be modified and used to talk to multiple Identity Manager 7.0 installations or third-party resources that are supporting SPML 2.0 core operations.

NOTE This sample adapter is provided in the Sun Resource Extension Facility Kit on your product CD or install image (located in /REF).

Appendix A

Using the Business Process Editor

This appendix provides instructions for using the Business Process Editor (BPE). The information in this chapter is organized as follows:

Overview

The Business Process Editor (BPE) is a standalone, Swing-based Java application that provides a graphical and forms-based view of Sun Java™ System Identity Manager workflows, forms, rules, generic and configuration objects, and views.

You use the BPE to customize Identity Manager for your environment as follows:

- View, edit, and create forms, workflows, rules, email templates, and rule libraries
- View and edit configuration objects and generic objects
- View JavaDocs for the classes that comprise the Identity Manager public APIs
- Debug forms, workflows, and rules
- Create workspaces that are associated with specific repositories

Starting and Configuring the BPE

NOTE To run the BPE, you must have Identity Manager installed on your local system and Configurator-level access to Identity Manager.

This section provides instructions for starting and configuring the BPE, including:

- Starting the BPE
- Specifying a Workspace
- Enabling JDIC
- Using SSL in the BPE

Starting the BPE

To start the BPE from the command line:

1. Change to the Identity Manager installation directory.
2. Set environment variables with these commands:

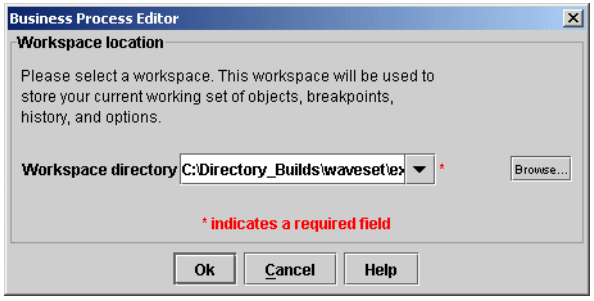
   ```
   set WSHOME=<Path_to_idm_directory>
   set JAVA_HOME=<path_to_jdk>
   ```

   To start the BPE on a UNIX system, you must also enter

   ```
   export WSHOME JAVA_HOME
   ```

3. Change to the `idm\bin` directory and type `lh config` to start the BPE. The Workspace location dialog displays, as shown in Figure A-1.

Figure A-1 BPE Workspace Location Dialog



Use the Workspace location dialog to create a new workspace or to select an existing workspace. Instructions for both actions are provided in the next section.

Specifying a Workspace

A *workspace* is a mechanism for saving repository connection information (such as the default server and password), options, breakpoints set by the BPE debugger, open sources, and automatically saved files.

A workspace is tied to a specific repository. You can have more than one workspace associated with a repository, but only one repository per workspace.

The BPE has two different connections to the Identity Manager repository:

- **Editor Connection** — This connection is used by the classic Editor portion of the BPE.

  The Editor can connect in the following ways:

  - **LOCAL**: The Editor connects the directory to the repository using the `ServerRepository.xml` in `WSHOME`.

    You can use the LOCAL connection to edit objects in the repository when the application server is not running.

  - **SOAP**: The Editor connects to the application server using SOAP.

- **Debugger Connection** — This connection is used by the Debugger portion of the BPE to:

  - Fetch source code from the application server
  - Receive the current debugging state (variables, current location)
  - Send commands to the debugger agent, which runs within the application server (setting breakpoints, sending step commands).

  Because sending commands to the debugger agent requires a connection to a live application server, the only valid setting for the Debugger connection is SOAP. If you choose SOAP for the Editor connection, the debugger will use the same connection as the editor.

This section contains instructions for

- Creating New Workspace
- Selecting a Workspace
- Troubleshooting Start-Up

Creating New Workspace

To create a new workspace, use the following instructions:

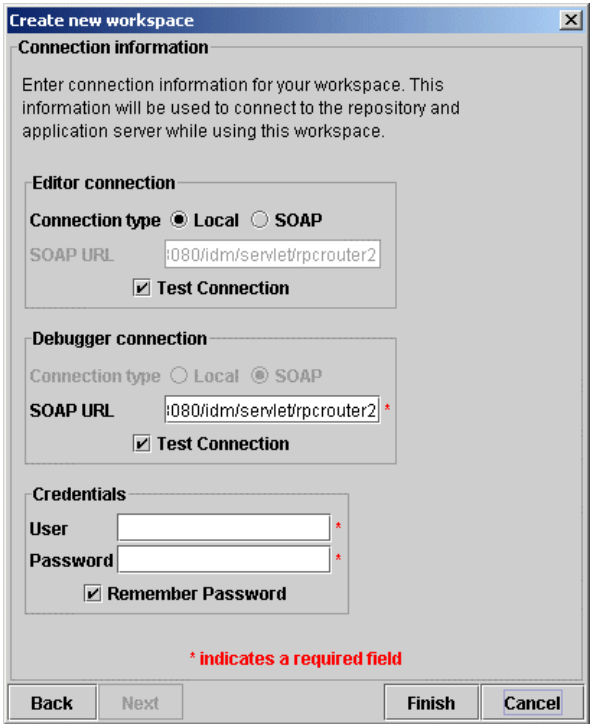
1. In the Workspace location dialog, enter a unique name for the new workspace in the Workspace Directory field, and then click OK.

   When you provide the name of a workspace that does not yet exist, the Create new workspace wizard displays and instructs you to provide a directory for the workspace.

2. Enter a directory name in the Workspace directory field, and then click Next.

   The Connection Information dialog displays so you can specify connection information for your workspace.

Figure A-2 BPE Connection Information Dialog



3. Specify the Editor connection information as follows:

   a. Select a connection type:

      - **Local** (selected by default): Select to enable the BPE to work on objects in a local repository.

        When you specify a Local connection, BPE connects to the repository using the `ServerRepository.xml` found in `WSHOME`. (The SOAP URL field will be greyed out.)

      - **SOAP**: Select to enable the BPE to work on objects in a different repository.

        When you specify a SOAP connection, you will also be specifying SOAP as the default connection type for the BPE debugger.

   b. If you are using a SOAP connection, enter a fully qualified URL in the SOAP URL field. For example, http://localhost:8080/<*idm*>/servlet/rpcrouter2, where <*idm*> is the directory where you installed Identity Manager.

   c. Enable the Test Connection option if you want Identity Manager to test this connection to the repository.

4. Specify the Debugger connection information for the BPE debugger as follows:

As mentioned previously, if you selected SOAP for the Editor connection type, you set the default debugger connection type to SOAP by default. All of the options in the Debugger connection area will be greyed out.

   a. Select the connection type and SOAP URL (if necessary).

   b. Enable the Test Connection option if you want Identity Manager to test this connection to the repository.

5. Provide the following credentials:

   a. Enter a User login name and Password.

   b. Enable the Remember Password option if you want the BPE to use these credentials by default whenever you log into BPE.

6. Click Finish to create the new workspace and the BPE main window displays.

Selecting a Workspace

Use one of the following methods to select an existing workspace from the Workspace location dialog,

- Select a workspace name from the Workspace directory menu list.
- Click Browse to locate and select a workspace.

After selecting a workspace, click OK and the BPE main window will display.

Troubleshooting Start-Up

When BPE tries to connect to the underlying server, you may receive the following error message:

```
HTTP 404 - /idm/servlet/rpcrouter2
Type Status report
message /idm/servlet/rpcrouter2 description
The requested resource (/idm/servlet/rpcrouter2) is not available
```

If you get this connection error, check the URL field in the browser instance in which you are running Identity Manager. The first part of the URL listed there — for example, http://localhost:8080/idm — must be the same as the URL that you entered as the debugger connection.

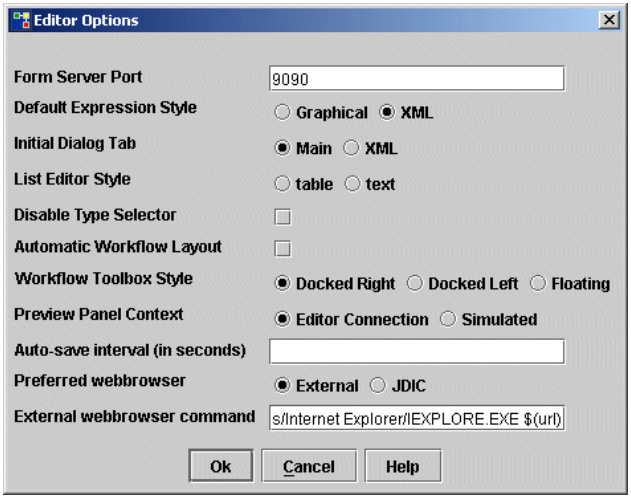
Enabling JDIC

If you want to embed the Web browser panel in the Form Preview panel, you must select JDIC as the preferred Web browser. Otherwise, the Web Browser panel will use the `External webbrowser` command to launch the Web browser externally.

To specify JDIC, use the following steps:

1. Select Tools > Options to open the Editor Options dialog.

Figure A-3 Editor Options Dialog



2. Enable the JDIC option as the Preferred webbrowser. (This option does not display if the application is running a version of JRE less than 1.4.)

NOTE

- To enable JDIC for Windows, you must install Internet Explorer. (Mozilla is not currently supported on Windows.)
- To enable JDIC on Linux or Solaris, you must install Mozilla. At present, GNOME is the only supported desktop.

  In addition, you must set the `MOZILLA_FIVE_HOME` environment variable to the root directory of your Mozilla installation.

To configure JDIC for Solaris 10+x86:

1. Download `jdic-0.9.1-bin-cross-platform.zip` from `https://jdic.dev.java.net`.
2. Extract the zipped files.
3. Replace the <*wshome*>`/WEB-INF/lib/jdic.jar` with the `jdic-0.9.1-bin-cross-platform/jdic.jar`.
4. Copy `jdic-0.9.1-bin-cross-platform/sunos/x86/*` to <*wshome*>`/bin/solaris/x86`.

Using SSL in the BPE

To use SSL. in the BPE, open the Create new workspace wizard and change the SOAP URL protocol to `https` and the port number to your application's SSL port.

Navigating the Business Process Editor

Before you start customizing Identity Manager processes or objects, you should know how to work with, view, and enter information and how to make selections in the BPE.

This information is organized into the following sections:

- Working with the BPE Interface
- Loading Processes or Objects
- Setting Editor Options
- Validating Workflow Revisions
- Saving Changes
- Inserting XPRESS
- Using Keyboard Shortcuts

Working with the BPE Interface

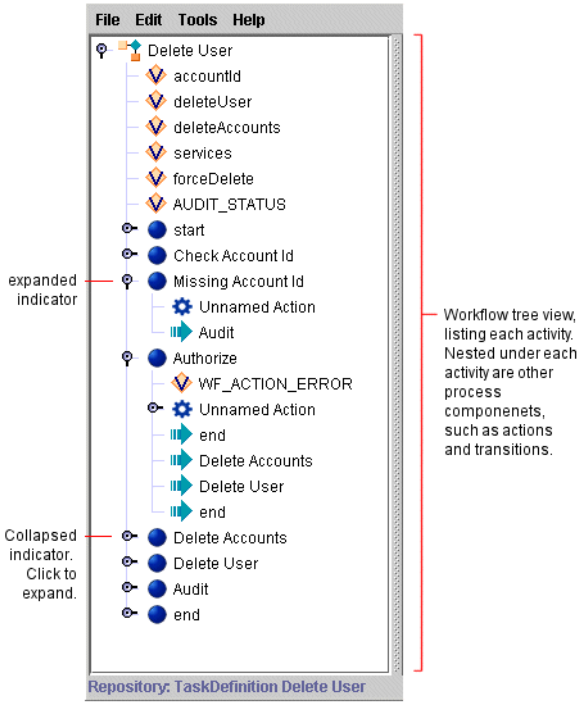
The BPE interface includes a menu bar and dialogs for selections. The primary display is divided into two main panes:

- Tree view
- Additional display views, which include
  - Diagram view
  - Graphical view
  - Property view

Working in Tree View

Tree view (in the left pane) provides a hierarchical view of tasks, forms, views, or rules. This view lists each variable, activity, and subprocess in order — nesting actions and transitions under each activity. Figure A-4 shows a sample tree view highlighting workflow.

Figure A-4 BPE Tree View



Working with Additional Display Views

BPE provides the following additional display views:

- Diagram View
- Graphical View
- Property View

The availability of these views depend upon the object type or process you select. For example, the BPE presents a graphical display of a form as it would appear in a browser. This view complements the property view and XML display of unique form elements.

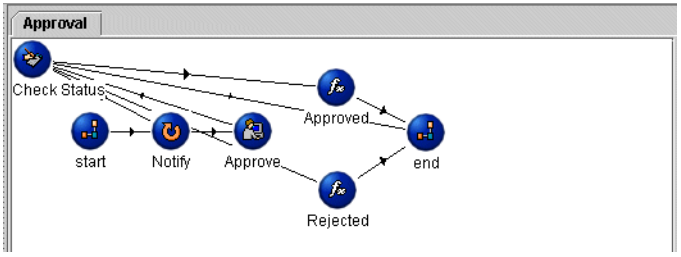
These views are introduced in the following sections.

NOTE For more detailed information about which display types are available for each Identity Manager object or workflow process, and how to work with these additional views, see *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

Diagram View

For workflow, the Diagram view displays in the right interface pane, and provides a graphical representation of a process. Each icon represents a particular process activity.

Figure A-5 Diagram View (Workflow)



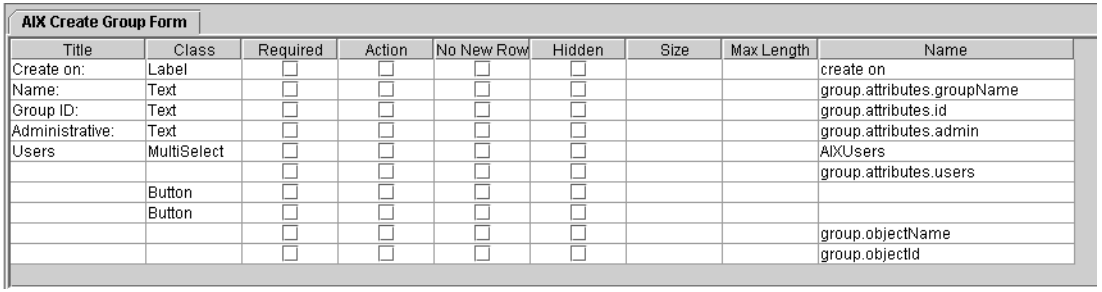
Graphical View

The Graphical view displays in the lower right pane of the BPE display and shows the currently selected form as seen in a browser window.

Property View

The Property view displays in the upper right pane of the BPE display and provides information about elements in the currently selected form.

Figure A-6 Property View (Form)



AIX Create Group Form

Title	Class	Required	Action	No New Row	Hidden	Size	Max Length	Name
Create on:	Label	☐	☐	☐	☐			create on
Name:	Text	☐	☐	☐	☐			group.attributes.groupName
Group ID:	Text	☐	☐	☐	☐			group.attributes.id
Administrative:	Text	☐	☐	☐	☐			group.attributes.admin
Users	MultiSelect	☐	☐	☐	☐			AIXUsers
		☐	☐	☐	☐			group.attributes.users
	Button	☐	☐	☐	☐			
	Button	☐	☐	☐	☐			
		☐	☐	☐	☐			group.objectName
		☐	☐	☐	☐			group.objectId

Loading Processes or Objects

To load an Identity Manager process or object use the following steps:

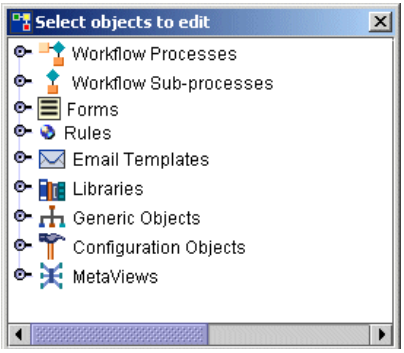
1. Select File > Open Repository Object from the menu bar.

   TIP You can also use the Ctrl-O shortcut. (See "Using Keyboard Shortcuts" on page 311, for a complete list of BPE shortcuts.)

2. If prompted, enter the Identity Manager Configurator name and password in the login dialog, and then click Login.

   The Select objects to edit dialog displays (similar to Figure A-7).

Figure A-7 Select objects to edit dialog (with Library options expanded)



This dialog contains a list of objects, which can include the following object types.

- Workflow Processes
- Libraries
- Workflow Sub-processes
- Generic Objects
- Forms
- Configuration Objects
- Rules
- MetaViews
- Email Templates

Items displayed may vary based on your Identity Manager implementation.

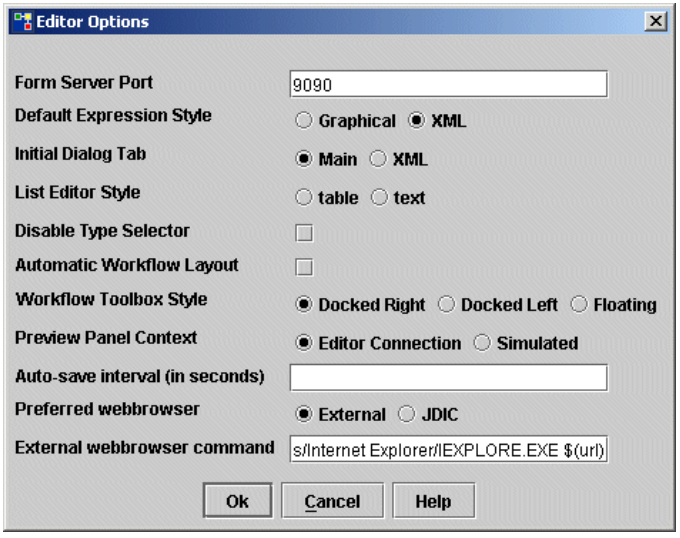
3. Double-click an object type to display all of the objects that you have permission to view for that type.
4. Select a process or object, and then click OK.

Setting Editor Options

You can set several options so that your preferences are reflected each time you launch the BPE. You can also set these options individually each time you work in the editor.

To set editor options, select Tools > Options to open the Editor Options dialog.

Figure A-8 Editor Options Dialog



You can use the options on this dialog to specify the following preferences:

- **Form Server Port** — Specifies the default port for the HTML Preview page. You use this page when you are editing forms.
- **Default Expression Style** — Controls the display option for expressions in forms, rules, and workflows (Graphical or XML).
- **Initial Dialog Tab** — Controls the tab that appears on top (Main or XML).
- **List Editor Style** — Controls the default display of list expressions. You can display lists in a table or as text boxes.
- **Disable Type Selector** — Disables the Type Selector option that appears next to text boxes. The option to change types will still be available through the Edit dialog.

- **Automatic Workflow Layout** — Enables automatic layout of workflow activities the first time it is opened.
- **Workflow Toolbox Style** — Specifies where the Workflow Toolbox will be displayed relative to the main BPE window. Options include
  - **Docked Right** (Default): Select to dock the toolbox to the right side of the BPE window.
  - **Docked Left**: Select to dock the toolbox to the left side of the BPE window.
  - **Floating**: Select if you want the ability to move the toolbox around the BPE window.
- **Preview Panel Context** — Identifies the context in which information displayed in the Preview pane is rendered. Options include
  - **Editor Connection** — Select if you want the BPE to always attempt to connect to the repository.
  - **Simulated** — Select to work on forms off-line.
- **Auto-save interval (in seconds)** — Specifies how many seconds the BPE will wait before auto-saving a session. (Default is 30 seconds)
- **Preferred webbrowser** — Specifies how to launch the Web browser. Options include:
  - **External** (Default): Select to have BPE use the `External webbrowser` command to launch the Web browser externally.
  - **JDIC**: Select to launch the Web browser panel in the Form Preview panel.
- **External webbrowser** command — Specifies the `External webbrowser` command to invoke the external Web browser.

Validating Workflow Revisions

You can validate your workflow revisions at different stages of the customization process:

- If you are working with XML display values, when adding or customizing variables, activities, actions, or transitions, click Validate to validate each change.
- After making changes, select the object or process in the tree view, and then select Tools > Validate to test it.

The BPE displays validation messages that indicate the status of the process:

- **Warning indicator (yellow dot)** – Indicates that the process action is valid, but that the syntax style is not optimal.
- **Error indicator (red dot)** – Indicates that the process will not run correctly. You must correct the process action.

Validate your workflow revisions as follows:

1. Click an indicator to display its process action.
2. After making changes, click Re-validate to re-test the process, confirm that the error is corrected, and check for additional errors.
3. Drag your cursor into the Workflow diagram view.

   The activity appears in the view.

TIP	Any activities you create after the first activity are numbered. Re-number similar activities before creating more than two.

Saving Changes

To save your changes to a process or object and check it into the repository, select File > Save in Repository from the menu bar. When you select Save, the BPE saves the object in the location in which it was last saved (either in the repository or the file in which it was last saved). You can have multiple copies of the same object open in varying states and in different files or repositories.

NOTE	You can also use File > Save As File to save the object or process as an XML text file. Save the file in the form *Filename*`.xml`.

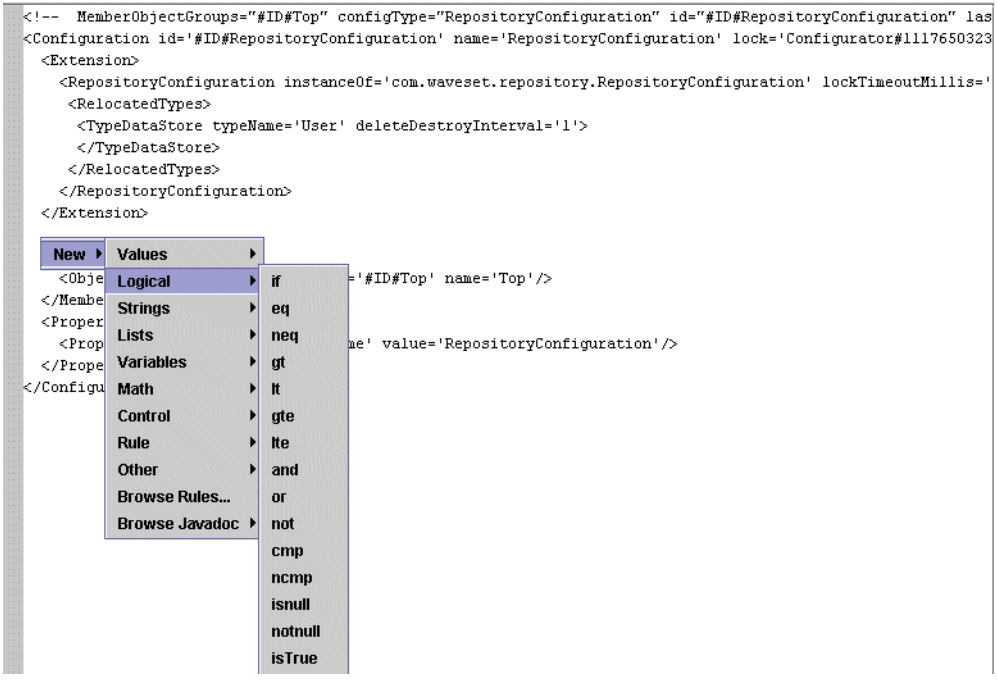
Inserting XPRESS

If you are editing a rule, workflow, configuration or generic object, or form in the BPE XML pane, you can quickly insert an XML template for an XPRESS element wherever you have positioned the cursor.

1. Position the cursor where you want to add the new XPRESS statement.
2. Click the right mouse button to display the New menu.
3. Select the type of XPRESS statement you want to add to the XML.

   For example, select New > Logical > cond to add an empty `cond` statement at the cursor insertion point. The BPE displays the content-free `cond` statement, as illustrated in the following figure.

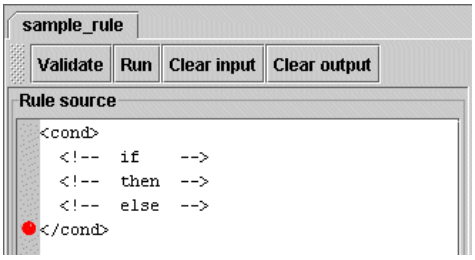
Figure A-9 Menu for Inserting XPRESS Functions into XML



4. Complete the statement as needed.

If you insert the XPRESS element in an invalid location, one or two red dots (indicators) display immediately to the left of the new code lines that mark the first and last lines of the inserted code. See Validating Workflow Revisions for information about these indicators.

Figure A-10 Inserting XPRESS Function



Using Keyboard Shortcuts

The BPE supports these keyboard shortcuts for performing tasks.

Table A-1 BPE Keyboard Shortcuts

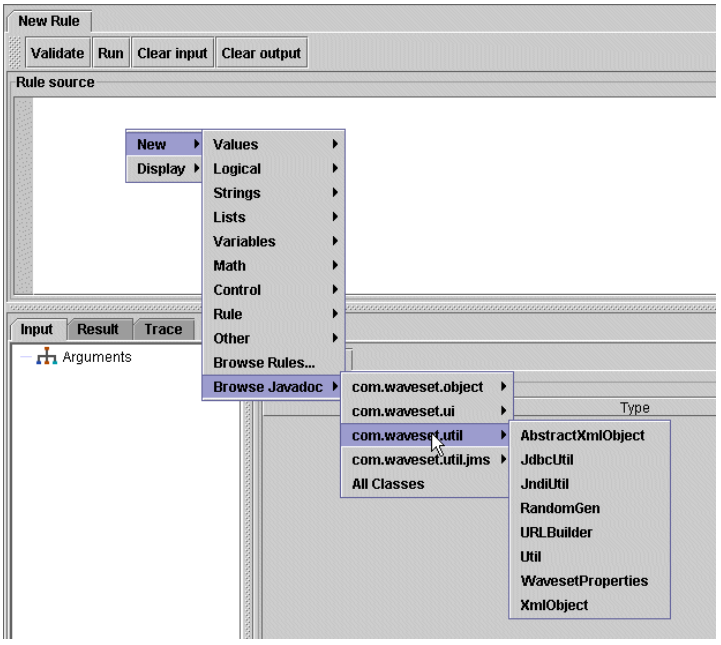
Keyboard Command/Key	Action
Ctrl-C	Copy
Ctrl-O	Open (repository object)
Ctrl-R	Refresh Sources
Ctrl-S	Save (repository object)
Ctrl-V	Paste
Ctrl-X	Cut
Delete	Delete
F5	Select Current Line
F6	Step out
F7	Step into
F8	Step over
F9	Continue (Debugging)

Accessing JavaDocs

You can access JavaDocs for all public method classes from any BPE window that displays XML, as follows:

1. Right-click in an XML window to display the cascading menu.
2. Select New > Browse Javadoc.

Figure A-11 Opening a Javadoc



3. Select one of the following options from the cascading menu, which includes the following packages, subsequently broken down into component classes:
   - **com.waveset.object**: Lists all the classes subordinate to this parent class.
   - **com.waveset.ui**: Lists all the classes subordinate to this parent class.
   - **com.waveset.util**: Lists all the classes subordinate to this parent class.
   - **com.waveset.util.jms**: Lists all the classes subordinate to this parent class.

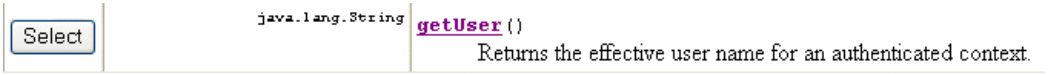
- **All Classes**: Displays the frames view of the JavaDoc classes, from which you can navigate through each class in the browser.

Selecting one of these menu options opens a browser window that displays the class `Javadoc`.

Inserting a Method Reference

To insert a method invocation in the XML, access the method summary section of the class `Javadoc`. Click the Select button that precedes the method name under the Method Summary.

Figure A-12 Selecting the getUser Method



At the cursor insertion point, BPE inserts the `<invoke>` element that you need to call the method from the XML.

TIP Click **Validate** for preliminary confirmation of the invoke statement syntax and XML.

Working with Generic and Configuration Objects

The fundamental object model for Identity Manager is the *persistent object model*. Because you perform almost all Identity Manager operations by creating an object, the persistent object API is the fundamental object model for customizing and controlling Lighthouse.

This section provides information about working with persistent objects. The information is organized as follows:

Common Persistent Object Classes

`PersistentObject` is the common base class of all persistent objects, and provides the fundamental object model for customizing and controlling Identity Manager. `PersistentObject` consists of a set of Java classes that are part of the infrastructure that is common to all persistent objects.

These common `PersistentObject` classes include:

- **Type**: A set of constants used in many methods to indicate the type of object being referenced.
- **PersistentObject**: The common base class of all repository objects. The most significant properties are the *identity*, *member object groups* and the *property list*.
- **ObjectRef**: When an object references another, the reference is encoded in this object. The reference includes the object type, name, and repository identifier.
- **Constants**: A collection of random constants for many different system components.
- **ObjectGroup**: A group representing an *organization* in the Identity Manager interface. All persistent objects must belong to at least one object group. If you do not specify otherwise, the object is placed in the Top group.
- **Attribute**: A collection of constant objects that represents common attributes that are supported by objects. Often used internally when building object queries. When a method accepts an Attribute argument, there is usually a corresponding method that takes a string containing the attribute name.

Viewing and Editing Objects

You can use the BPE to view and edit the two of the most commonly customized persistent object types:

- **Configuration objects**: A type of persistent object that contains forms and workflow processes.
- **Generic objects**: A configuration object that has an <*Extension*> of type <*Object*>. (In contrast to workflows, which are Configuration objects that have an <*Extension*> of type <*WFProcess*>.) You typically use Generic objects to represent views, and they are simple collections of name/value pairs. You can access these attributes externally through path expressions.

The following sections provide an introduction to the Configuration and Generic object types. For more detailed information, see *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

Configuration Objects

You can directly access forms and workflows in the BPE; however, the BPE also provides access to other configuration objects that are not associated with a custom viewer. You can access these miscellaneous configuration objects from the BPE under the Configuration Object category.

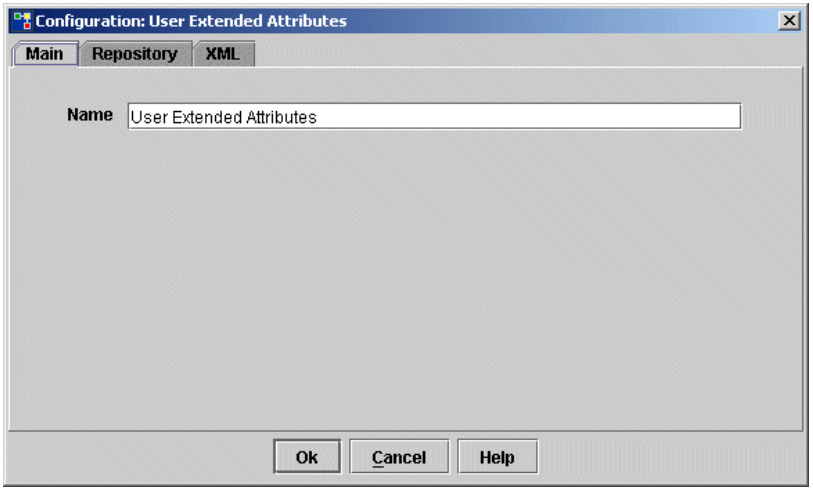
The BPE lists these miscellaneous configuration objects in the left pane (tree view), as shown in the following figure:

Figure A-13 BPE Tree Display of the Configuration Object

Double-clicking on an object name in tree view displays the Object window, which provides the following object views (tabs): Main, Repository, and XML.

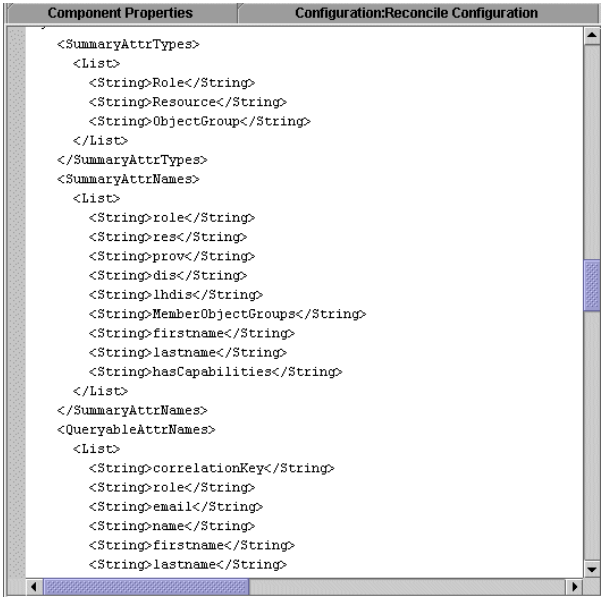
For example, if you double-click User Extended Attributes in tree view, the following dialog displays:

Figure A-14 User Extended Attributes Object Dialog



The BPE also displays configuration objects as unfiltered XML in the left pane of the BPE window. For example, see the following figure:

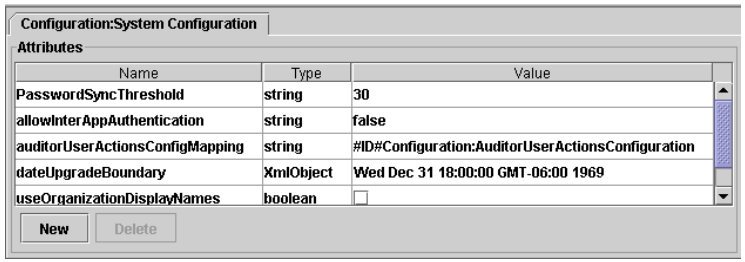
Figure A-15 BPE XML Display of Reconcile Configuration Object



Generic Objects

Generic objects are simple collections of name/value pairs you can use to represent views. BPE displays these name/value pairs in column form and lists the attribute's data type. Valid data types include Boolean, int, string, and xmlobject.

Figure A-16 BPE Attribute Display of Generic Object (System Configuration)



Many customizations involve editing the System Configuration object, which is a type of generic object.

Creating a New Object

To create a new Configuration or Generic object

1. Select File > New > GenericObject or Configuration Object.

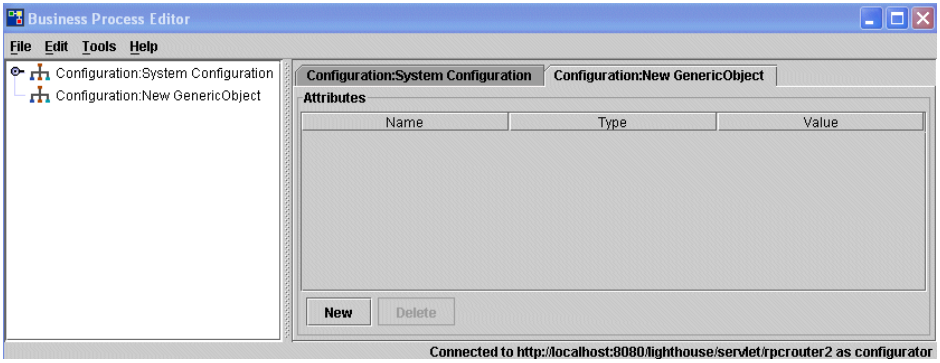
   The Configuration:New GenericObject or Configuration:New Configuration dialog opens, with the Main panel displayed.

2. Enter the new object name in the Name field.

   The BPE main window adds the new object name to the tree view. In addition,

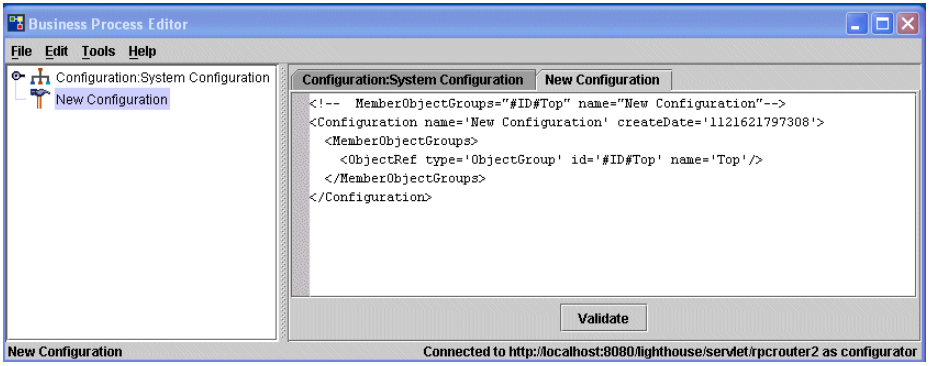
   - If you created a generic object, a blank Attributes pane displays as follows:

Figure A-17 BPE New Generic Object Display



   - If you created a configuration object, the BPE displays the following window, which contains a template for the new XML object.

Figure A-18 BPE New Configuration Object Display

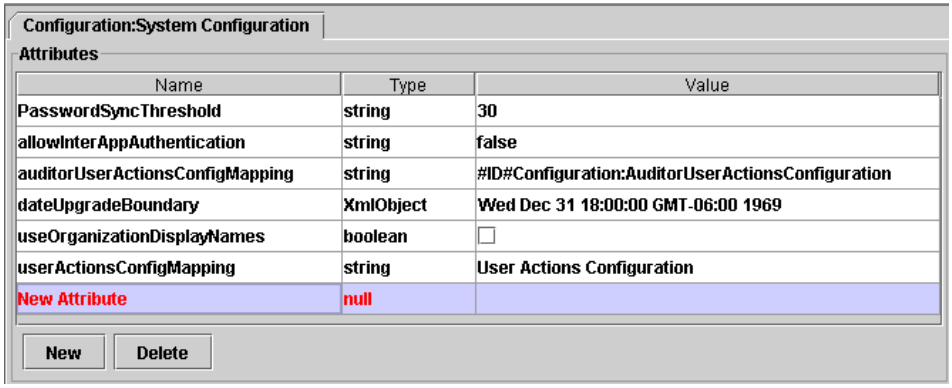


3. If you are creating a generic object, add an attribute as follows, repeating as necessary:

   a. Click New at the bottom of the Attributes pane. The BPE displays a new attribute field at the bottom of the list of attributes. Select New Attribute, then enter the name of the attribute.

   b. Assign a data type by clicking null in the Type column, and selecting a data type from the drop-down menu.

Figure A-19 New Attribute of BPE Generic Object Display



NOTE To delete an attribute, click the attribute name, and then click Delete.

4. Select File > Save in Repository to save the new object to the repository.

Validating a New Configuration Object

You can immediately validate the new configuration object XML by clicking Validate, in the right pane of the main BPE window.

Creating and Editing Rules

You can use the BPE to

- View, create, and edit rules
- Test rules with a Lighthouse context
- Define data passed into the rule
- Save rule definitions to a file
- Retrieve information about a selected rule such as attribute types
- Display view attributes for reference while you customize rules

This section provides information and instructions for using the BPE to create and edit rules. The information is organized as follows:

NOTE Instructions for starting the BPE application are provided in "Starting and Configuring the BPE" on page 296.

Using the BPE Interface

Before you start customizing rules, you must understand the basics of navigating and using the BPE interface. When you are working with rules, the initial BPE interface consists of display panes, a menu bar, Action menus, and Rule dialogs.

NOTE The BPE interface changes based on the object type or process selection.

This section describes the interface related to creating and editing rules. The information is organized as follows:

BPE Display Panes

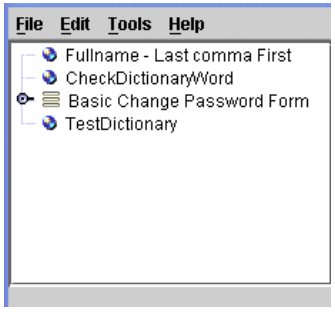
When you are working with rules, the BPE interface provides the following display panes:

- Tree view
- Rule source
- Input tab
- Trace tab
- Result tab

Tree View

The tree view (in the left interface pane) lists selected rules as standalone icons.

Figure A-20 Rule Display in Tree View



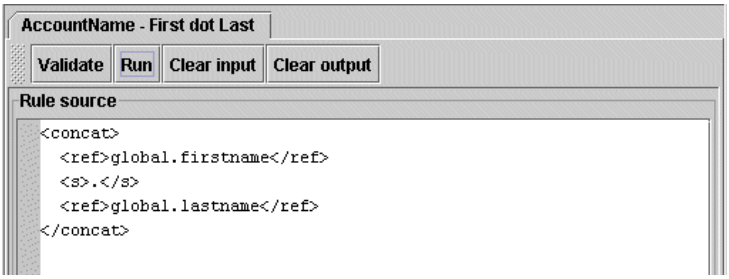
Typically, the tree view shows a hierarchical view of tasks, forms, or views — listing each element in order with sub-elements nested under their parent.

However, rules do not reside in hierarchies within Identity Manager (unless already incorporated into a rule library object, workflow, or form), so there are no hierarchical relationships among rules to display in tree view. Instead, rules that are not incorporated into a library, workflow, or form appear in tree view as single icons.

Rule Source

The Rule source pane (in the upper right interface pane) provides the source information of a rule.

Figure A-21 Rule Source Pane



From this pane, you can right-click to access a cascading menu that enables you to perform any of the following tasks:

- Create a new rule or add new values to the selected rule
- Browse and select existing rules and libraries
- Browse and view existing JavaDocs
- Change the display to view the rule source in XML, Graphical, Property Sheet, or Configuration format.

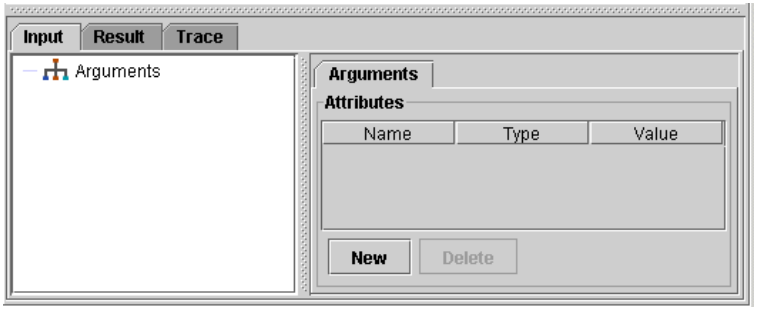
You can also use the buttons located above this pane to perform any of the following actions:

- **Validate**: Validates the rule with the current set of arguments
- **Run**: Executes the rule with the current set of arguments
- **Clear the input**: Resets the input arguments to their defaults
- **Clear the output**: Clears the Result and Trace panes

Input Tab

The Input tab pane (located in the lower right corner of the window) displays by default.

Figure A-22 Input Tab Pane



You can use this tab to control the arguments that are passed to the rule for testing. This tab is basically the same as the BPE's GenericObject Editor (see "Generic Objects" on page 316).

From this pane, you can:

- Double-click the argument name, the Arguments dialog displays so you can Validate the argument.
- Right-click an argument name to access a cascading menu that enables you to import test data from a view or a file. Specifically, you can perform any of the following tasks:
  - Insert the argument into a List, GenericObject, Map, or Test data
  - Edit the argument
  - Copy the argument
  - Paste the copied argument to another location
  - Import test data from a file
  - Export test data to a file
- Click New to create new arguments by specifying a name, type, and value.
- Click Delete to delete a selected argument.

Result Tab

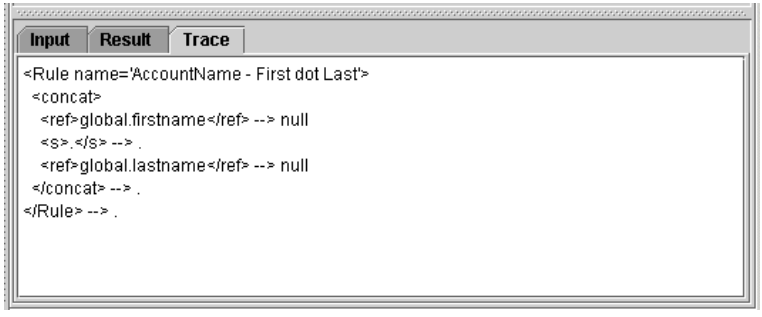
Select the Result tab and click Run (above the Rule source pane) to execute the selected rule. The rule's return value displays in the Result tab pane in XML format.

Figure A-23 Result Tab Pane

Trace Tab

Select the Trace tab to capture XPRESS tracing during execution of the rule.

Figure A-24 Trace Tab Pane



Menu Selections

You can use the menu bar or the action (right-click) menu to work in the interface.

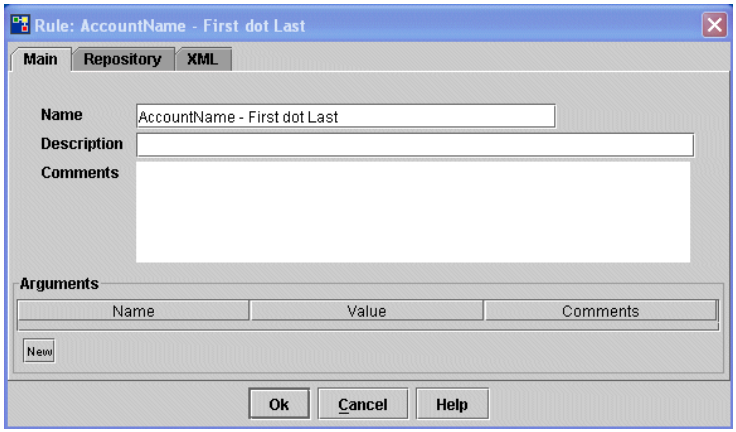
Select an item in the tree or diagram view, and then right-click to display the action menu selections that are available for that item.

Rule Dialogs

Each rule and rule element has an associated dialog that you can use to define the element type and its characteristics.

To access these dialogs, double-click the rule name in the tree view. The rule dialog for the selected rule displays, with the Main tab in front by default. For example, see the following figure:

Figure A-25 Rule Dialog (Main Tab View)



You use the options on this dialog to define a rule, as follows:

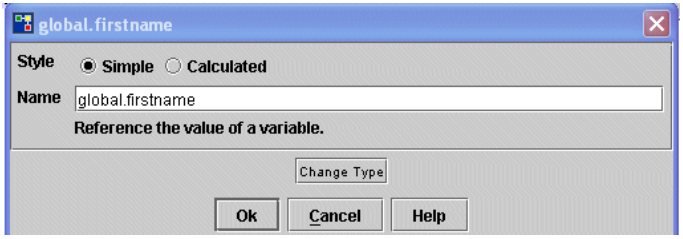
- **Name**: Automatically displays the selected rule name, which is the name displayed in the Identity Manager interface.
- **Description** (optional): Specify text describing the purpose of the rule.
- **Comments**: Specify text inserted into the rule body with a `<Comment>` element.
- **Arguments**: Specify any required arguments.

Editing Rule Elements (Changing Field Value Type)

Some dialog fields behave differently depending on the field value type you selected.

- If the value type is `String`, you can type text directly into the field.
- If the value type is `Expression`, `Rule`, or `Reference`, click Edit to edit the value.

Figure A-26 Rule Argument Dialog



You can use one of the following methods to change a value type:

- Click Edit, and then click Change Type (if the current value is `String`).
- Right-click to access the actions menu, then select Change Type (if the current value is `Expression`, `Rule`, or `Reference`).

Changing Display Type

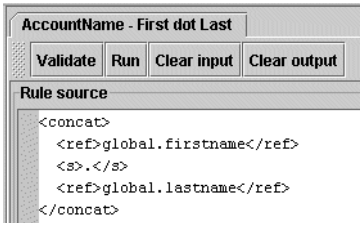
To change the way information displays in diagram view

1. Right-click to display the action menu.
2. Select Display > <*view_type*>.

   The view types include:

   - **XML** – Displays XPRESS or JavaScript source. You may prefer this display type if you are comfortable with XML.

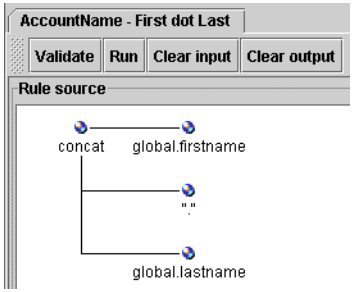
Figure A-27 XML Display



- **Graphical** – Displays a tree of expression nodes. This display type provides a structural overview.

NOTE To conserve space, Figure A-28 shows only part of the selected rule.

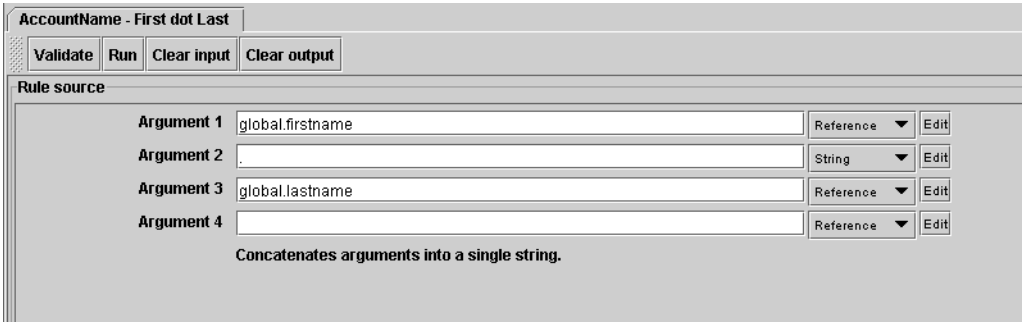
Figure A-28 Graphical Display



- **Property Sheet** – Displays a list of properties, some of which can be edited directly.

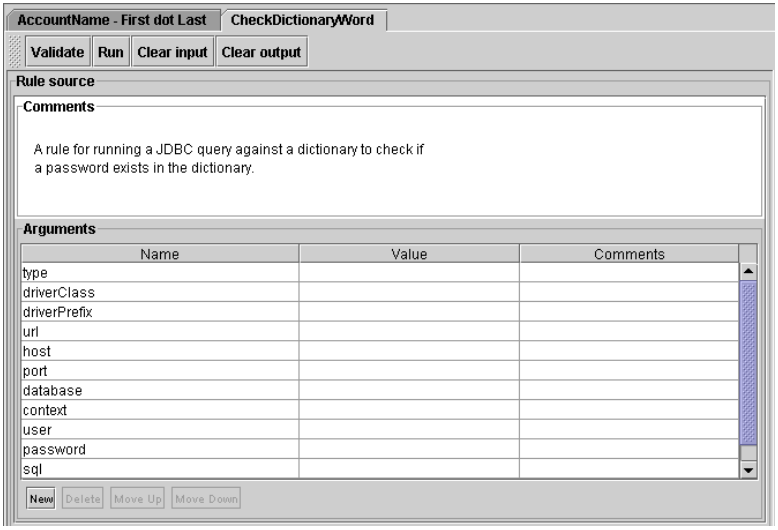
  You may be required to launch another dialog for other properties. For efficiency, use the Property Sheet display type when creating new expressions. (You can enter expression arguments more rapidly in this view compared to using the graphical view.)

Figure A-29 Property Sheet Display



- **Configuration** – Displays argument information listed in a property-sheet style. (See Figure A-30.) Also lists any comments that the rule creator used to describe the rule in the database.

Figure A-30 Configuration Display



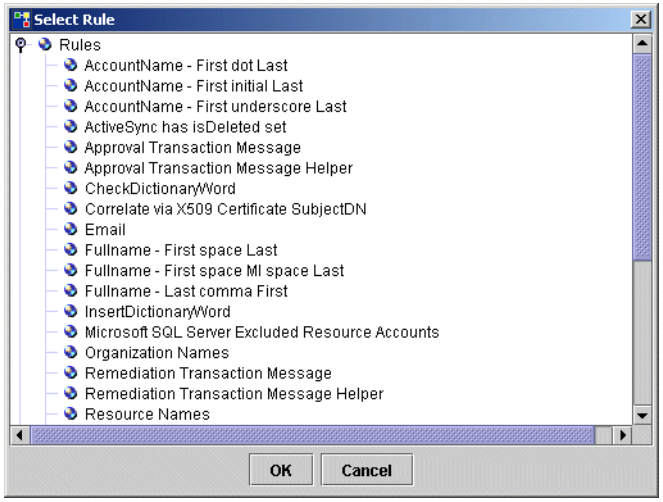
Browsing Rules

Use one of the following methods to browse and select the rules that you can access through Identity Manager:

- Select File > New Repository Object from the main menu bar. When the Select objects to edit dialog displays, expand the Rules node to display the available rules.
- Right-click in the Rule source pane and select New > Browse Rules from the actions menu.

When the Select Rule dialog opens (Figure A-31), expand the Rules node to display the available rules.

Figure A-31 Select Rule Dialog



Reviewing Rule Summary Details

Double-click a rule name in the tree pane to view, at a glance, rule elements. The Rule dialog contains the following tabs:

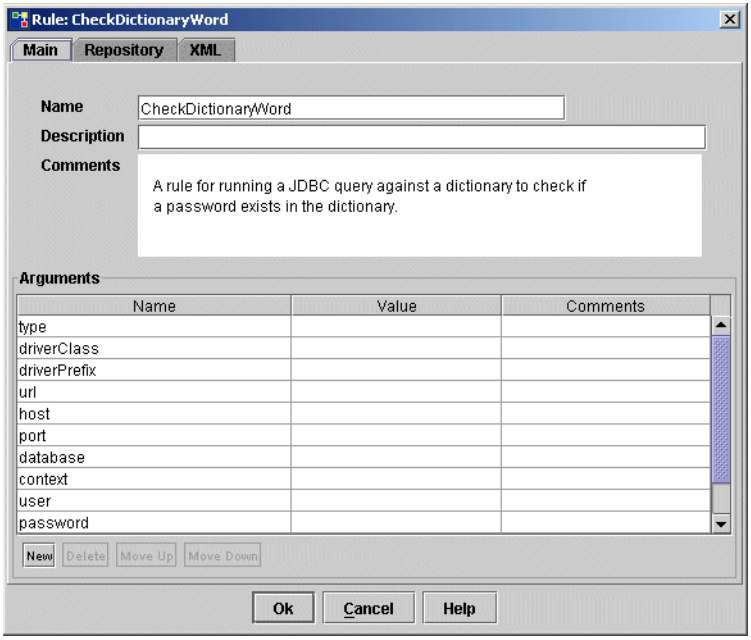
- Main
- Repository
- XML

Main Tab

Select this tab to access argument properties for this element (including each argument's name and value). It also lets you re-order arguments for better visual organization. (Reordering in this list does not change interpretation of the rule.)

The Main tab displays the same information about the rule as the Main view of the Rule dialog (see Figure A-32).

Figure A-32 Main Tab Display



Repository Tab

NOTE Rules that are not included in a Rule library have a Repository tab.

Select the Repository tab to view the following information about the selected rule:

Figure A-33 Repository Tab Display

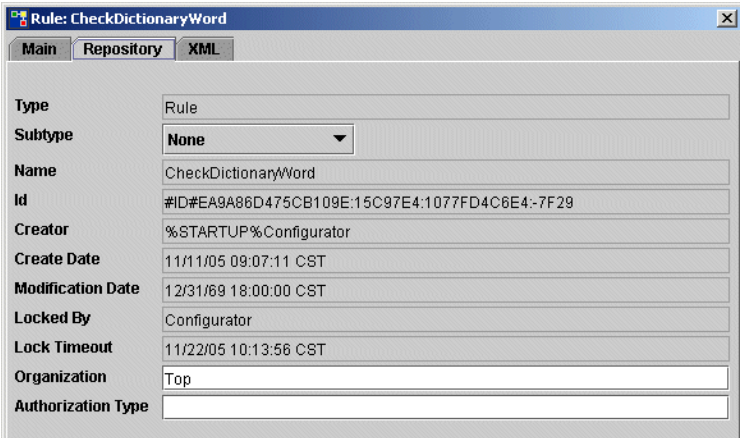


Table A-2 Fields on the Repository Tab

Field	Description
Type	Identifies the type of repository object. This value is always Rule.
Subtype	Identifies a subtype, if relevant. Rule subtypes are currently implemented within the Reconciliation interface only. The default is None.
Name	Assigned in the Rule dialog name field.
Id	Identification number assigned by Identity Manager.
Creator	Lists the account by which the rule was created.
CreateDate	Date assigned by Identity Manager when the object was created.
Modification Date	Date on which the object was last modified.
Organization	Identifies the organization in which the rule is stored.
Authorization Type	(Optional) Grants finer-grain permissions for users who do not have Admin privileges. The EndUserRule authorization type, for example, grants a user the ability to call a rule from a form in the Identity Manager User Interface.

The Repository tab primarily contains read-only information, but you can change the following values:

- **Subtype**: Select a new subtype assignment from the menu.

  Rules have no subtype by default. Consequently; when you create a rule, the Subtype value defaults to None.

  However, to display this rule in the Reconciliation interface, you must set this value to Account Correlation or Account Confirmation, depending upon which selection list in the Reconciliation graphical user interface you would like this rule to appear.

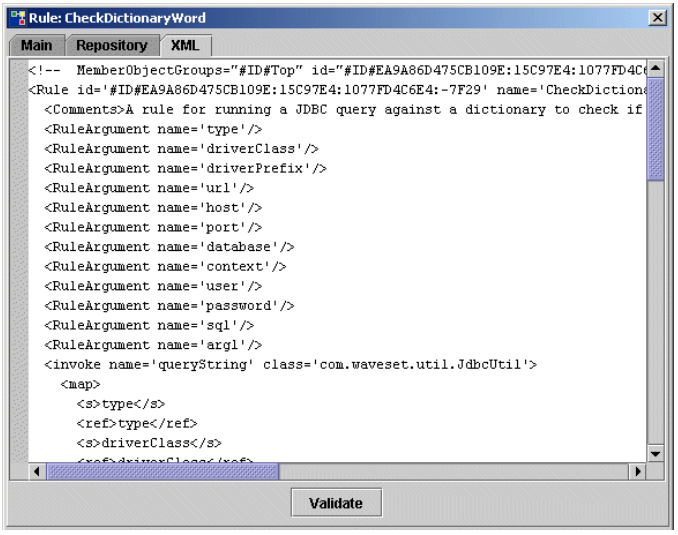
- **Organization**: Enter a new organization assignment into the text field.
- **Authorization Type**: Enter a new authorization type into the text field.

NOTE For an example rule subtype, see "Excluded Resource Accounts Rule SubType" on page 133.

XML Tab

Select the XML tab to view and edit raw XML for the selected rule. You can then click Validate to validate your changes before clicking OK to save. The XML parser validates the rule XML against the `waveset.dtd`.

Figure A-34 XML Tab Display

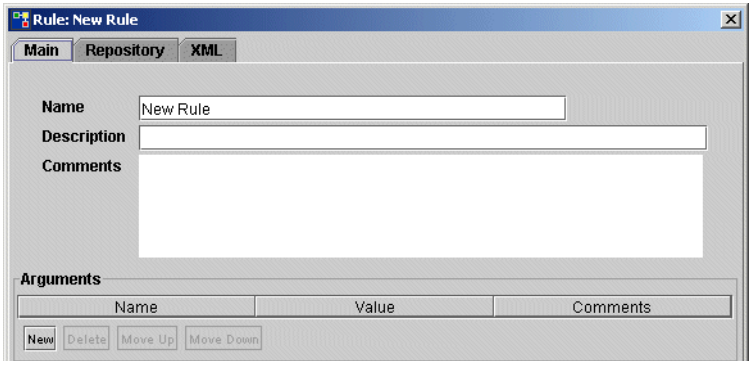


Creating a New Rule

Use the following steps to create a new rule:

1. Select File > New > Rule. and the Rule: New Rule dialog displays, with the Main tab in front by default.

Figure A-35 New Rule Dialog



2. Specify the following parameters for the new rule:
   - **Name** — Enter a name for the rule. (This name is displayed in the Identity Manager interface.)
   - **Description** (*Optional*) — Enter text to describe the purpose of the rule.
   - **Comments** — Enter text to be inserted into the rule body with a `<Comment>` element.
3. Click New to add arguments to the new rule.
4. When the Argument: Null dialog displays, enter text into the Name, Value, and Comments fields, and then click OK.

   This text displays in the Arguments table and will be inserted into the rule as a `<RuleArgument>` element.
5. When you are finished click OK to save your changes.

NOTE
- To remove an argument, click Delete.
- To change the arguments location in the Arguments table, click Move Up or Move Down.

Defining Rule Elements

The XML elements that comprise rules can be functions, XPRESS statements, one of several data types. You can use the following BPE Rule Element dialogs to create or edit rule elements:

- **Argument dialog** — Use to view or define argument characteristics.
- **Element dialog** — Use to view or define selected elements.
- **Object Access dialog** — Use to manipulate objects or call Java methods that manipulate objects.
- **Diagnostics dialog** — Use to debug or examine JavaScript, trace, print, and breakpoints.

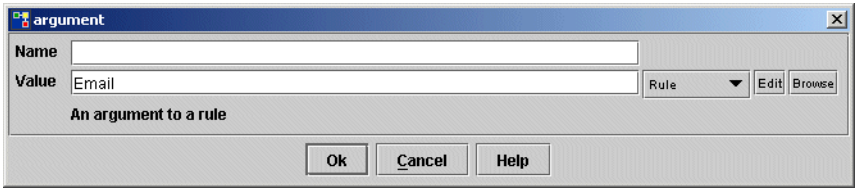
Additional information about these dialogs is provided in the following sections.

NOTE For more information about rule structure, see "Understanding Rule Syntax" on page 139.

Argument Dialogs

You use an Argument dialog to access and define rule arguments.

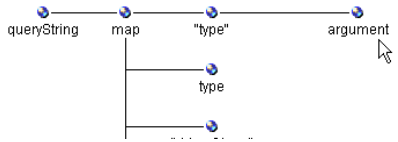
Figure A-36 Argument Dialog



Use one of the following methods to open an Argument dialog:

- From the Tree view pane, double-click a rule name to open the Rule dialog, and then double-click the argument name on the Main tab.
- From the Rule source pane (in Graphical View only), right-click and select New > Rule > argument.
- From the Rule source pane (in Graphical view only), double-click an argument node.

Figure A-37 Double-Click an Argument Node



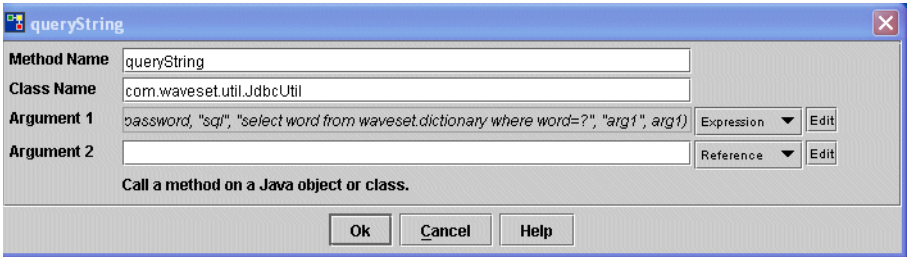
The Argument dialog provides the following basic options:

- **Name** — Specify a name for the argument. You can change the name from this dialog.
- **Value** — Specify the value of the selected argument.
- **Comments** — Specify optional comments.

In addition to the preceding options, the Argument dialog may also display other fields depending upon the type of element you are viewing/editing.

For example, if you are viewing a method element, an Argument dialog similar to the one displayed for the Query method dialog opens:

Figure A-38 Argument Popup Dialog (Method)



You can change the argument's data type by clicking the Change Type button, which displays the Select Type dialog.

Figure A-39 Select Type Dialog

Valid argument types are listed in the following table.

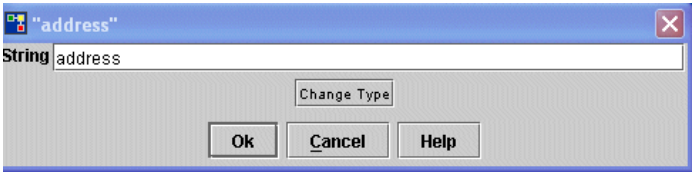
Table A-3 Valid Argument Types

Data Type	Description
String	Simple string constant.
Reference	Simple reference to a variable.
Rule	Simple reference to a rule.
List	Static list, such as an XML object list. This type is infrequently used in workflows (but used occasionally in forms).
Expression	Complex expression.
Map	Static map, such as an XML object map. Used rarely.
Integer	Integer constant. Can be used to make clearer the semantics of a value. Can be specified as String; the BPE coerces the string into the correct type.
Boolean	Boolean constant. Can be used to make clearer the semantics of a value. Can be specified as String. The BPE coerces the string into the correct type. Boolean values are specified with the strings `true` and `false`.
XML Object	Complex object that allows you to specify any of a number of complex objects with an XML representation. Some examples include EncryptedData, Date, Message, TimePeriod, and WavesetResult.

Element Dialogs

The Element dialog displays the name and value of an argument.

Figure A-40 Element Popup for the address Variable



Use one of the following methods to display an Element dialog from the Rule source pane (in Graphical view only):

- Double-click an element icon
- Right-click an element icon and select Edit from the action menu.

If you open a dialog by clicking the argument name, you can change the argument's data type and style (simple or calculated).

You can define several types of elements (see Table A-4). To change the data type of the element, click the Change Type button. The Select Type popup opens, displaying a list of the data types you can assign to the selected rule element.

To create a new element, right-click in the Graphical view and select New > <*element_type*> from the menu. The element types listed on this menu represent categories of XPRESS functions.

Table A-4 Element Types Representing XPRESS Function Categories

Menu Options	XPRESS Functions/Additional Actions You Can Invoke...
Values	`string`, `integer`, `list`, `map`, `message`, `null`
Logical	`if`, `eq`, `neq`, `gt`, `lt`, `gte`, `lte`, `and`, `or`, `not`, `cmp`, `ncmp`, `isnull`, `notnull`, `isTrue`, `isFalse`
String	`concat`, `substr`, `upcase`, `downcase`, `indexOf`, `match`, `length`, `split`, `trim`, `ltrim`, `rtrim`, `ztrim`, `pad`
Lists	`list`, `map`, `get`, `set`, `append`, `appendAll`, `contains`, `containsAny`, `containsAll`, `insert`, `remove`, `removeAll`, `filterdup`, `filternull`, `length`, `indexOf`
Variables	• Define a variable • Create a reference • Assign a value to a variable or an attribute of an object

Table A-4 Element Types Representing XPRESS Function Categories *(Continued)*

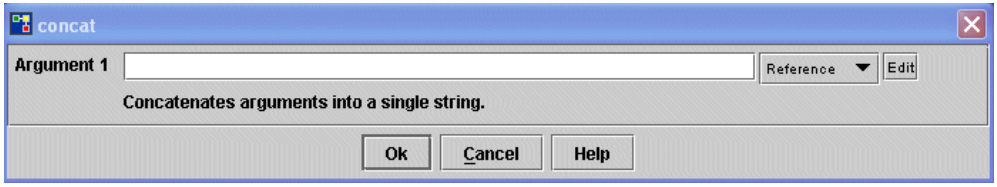
Menu Options	XPRESS Functions/Additional Actions You Can Invoke...
Math	`add`, `sub`, `mult`, `div`, `mod`
Control	`switch`, `case`, `break`, `while`, `dolist`, `block`
Rule	• Create new rule • Create argument
Other (functions, object access, diagnostics)	Displays further options: • Functions includes define function, define argument, and call function • Object access includes the `new`, `invoke`, `getobject`, `get`, and `set` functions. • Diagnostic includes options for creating or invoking Javascript, trace, print, and breakpoint functions.

NOTE For more information about these functions, see *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

You can also access the element types most recently created in a BPE session through the Recent options of the actions menu.

The following figure shows the window that displays when you select New > Strings > concat.

Figure A-41 concat Dialog



Object Access Dialogs

Use an Object Access dialog to manipulate objects or call Java methods that manipulate objects.

To open an Object Access dialog, right-click anywhere in the graphical display and select New > Other > Object Access > *option* from the pop-up menu.

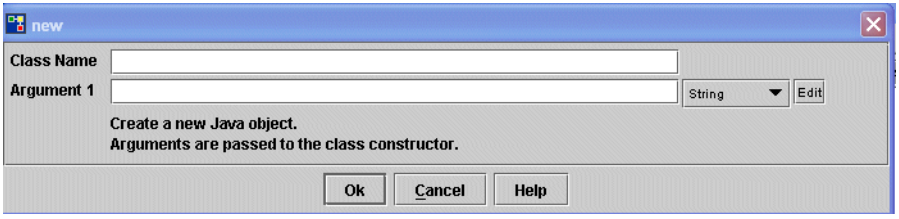
Where *option* can be any of the following options described in Table A-5:

Table A-5 Object Access Options

Option:	Description
new	Creates a new Java object. Arguments are passed to the class constructor
invoke	Displays the invoke dialog. Use to invoke a Java method on a Java object or class
getobj	Displays the getobj dialog. Use to retrieve an object from the repository
get	Retrieves a value from within an object. The first argument must be a List, GenericObject, or Object, and the second argument must be a String or Integer. • If the first argument is a List, the second argument is coerced to an integer and used as a list index. • If the first argument is a GenericObject, the second argument is coerced to a String and then used as a path expression. • If the first argument is any other object, the second argument is assumed to be the name of a JavaBean property.
set	Assigns a value to a variable or an attribute of an object.

To create an object, right-click to access the action menu, and select New > Other > Object Access > new.

Figure A-42 new Dialog

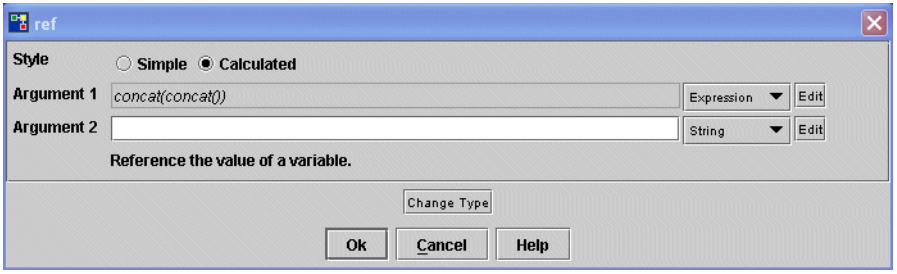


Editing Element Details

From the Argument dialog, you can define values for the variable. Use the Value field to enter a simple string value as the initial variable value. Alternatively, you can select a value type (such as Expression or Rule), and then click Edit to enter values.

Figure A-43 shows the variable window for a `ref` statement.

Figure A-43 ref Dialog



To identifying argument types (simple or complex), select `Simple` when you can enter the argument value in a text field (for example, string, Boolean, or integer). If you are working with a List, XML Object, or other expressions that requires an additional popup, select `Calculated`.

Diagnostics Dialogs

You can use Diagnostics dialogs to debug or examine the following:

- JavaScript
- Trace
- Print
- Breakpoint

To access the Diagnostics dialog, select New > Other > Diagnostics > trace from the actions menu in the right pane. Select an option described in Table A-6 to debug that item.

Table A-6 Trace Options

Option	Description
JavaScript	Displays the Script dialog so you can enter your own JavaScript.
trace	Inserts a `<trace>` XPRESS function into the rule. This function turns XPRESS trace on or off when this rule is evaluated. Set to `true` to enable trace or `false` (or just null) to disable.
print	Displays the Print dialog so you can enter the name of `tan` argument. This function is similar to the `block` function in that it contains any number of expressions and returns the result of the last expression. Enter an argument name in the **Argument** field, and select the type from the menu located next to the field. Default is `String`.
breakpoint	Displays the Breakpoint popup. Click **OK** to raise a debugging breakpoint.

Editing a Rule

If you customize a rule, you must save and validate your changes to ensure that the rule completes correctly and as expected. After saving, import the modified rule for use in Identity Manager.

This section provides instructions for the following:

- Loading a Rule
- Saving Changes
- Validating Workflow Revisions

Loading a Rule

Use the following steps to load a rule in the BPE:

1. Select File > Open Repository Object from the menu bar.
2. If prompted, enter the Identity Manager Configurator name and password in the displayed login dialog, and then click Login.

The following items display:

- Workflow Processes
- Workflow Sub-processes
- Forms
- Rules
- Email Templates
- Libraries
- Generic Objects
- Configuration Objects
- MetaViews

NOTE Items displayed may vary for your Identity Manager implementation.

3. Expand the Rule node to view all existing rules.
4. Select the rule you want to load, and then click OK.

NOTE If you are loading a rule for the first time, the rule components displayed in the right pane may not display correctly. Right-click in the right pane and select Layout to re-display the diagram.

Saving Changes

To save changes to a rule and check it into the repository, select File > Save in Repository from the menu bar.

NOTE You can also use File > Save As File to save the rule as an XML text file. Save the file in the form *Filename*`.xml`.

Validating Changes

You can validate changes to rules at different stages of the customization process:

- From the Rule source pane, click the Validate button to validate the rule with the current set of arguments.
- If you are working with XML display values, when adding or customizing arguments, click Validate to validate each change to the rule.
- After making changes, select the rule in the tree view, and then select Tools > Validate to test it.

The BPE displays validation messages that indicate the status of the rule:

- **Warning indicator (yellow dot)** — Indicates that the process action is valid, but that the syntax style is not optimal.
- **Error indicator (red dot)** — Indicates that the process will not run correctly. You must correct the process action.

Rule Libraries

A *rule library* serves as a convenient way to organize closely related rules into a single object in the Identity Manager repository. Using libraries can ease rule maintenance by reducing the number of objects in the repository and making it easier for form and workflow designers to identify and call useful rules.

A rule library is defined as an XML Configuration object. The Configuration object contains a Library object, which in turn contains one or more Rule objects. Code Example A-1 shows a library containing two different account ID generation rules:

Code Example A-1 Library with Two Account ID Generation Rules

```
<Configuration name='Account ID Rules'>
   <Extension>
      <Library>
         <Rule name='First Initial Last'>
            <expression>
               <concat>
                  <substr>
                     <ref>firstname</ref>
                     <i>0</i>
                     <i>1</i>
                  </substr>
                  <ref>lastname</ref>
               </concat>
            </expression>
```

Code Example A-1 Library with Two Account ID Generation Rules *(Continued)*

```
         </Rule>
         <Rule name='First Dot Last'>
            <expression>
               <concat>
                  <ref>firstname</ref>
                  <s>.</s>
                  <ref>lastname</ref>
               </concat>
            </expression>
         </Rule>
      </Library>
   </Extension>
</Configuration>
```

You reference rules in a library using an XPRESS `<rule>` expression. The value of the name attribute is formed by combining the name of the Configuration object containing the library, followed by a colon, followed by the name of a rule within the library.

For example, the following expression calls the rule named `First Dot Last` contained in a library named `Account ID Rules`:

```
<rule name='Account ID Rules:First Dot Last'/>
```

Selecting a Library to View or Customize

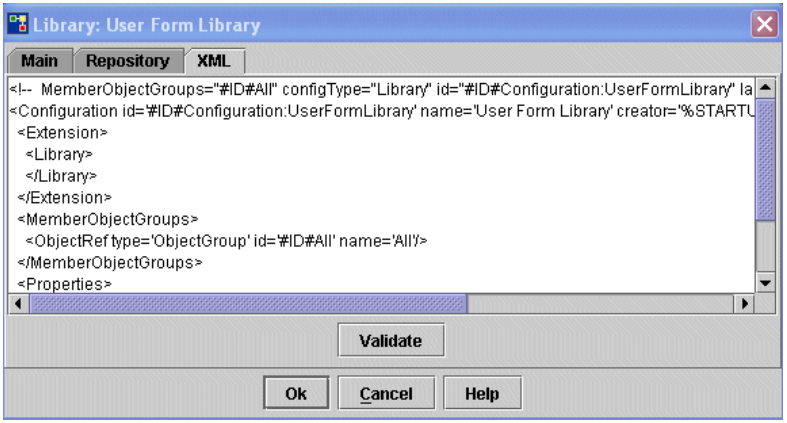
Perform the following steps to select a rule library to view or edit:

1. From the Business Process Editor, select File > Open Repository Object.

   Rule libraries are represented in the BPE with this icon  :

2. Select the rule library object in the Tree view, and select Edit.
3. Right-click inside the right edit pane, and then select the XML tab.

Figure A-44 Rule Library (XML View)



You can now edit the rule library XML.

Adding a Rule to an Existing Library Object

Once a rule library has been checked out, you can add a new rule by inserting the `<Rule>` element somewhere within the `<Library>` element. The position of the Rule within the library is not significant.

Customizing a Workflow Process

This section uses an Email Notification example to illustrate the end-to-end steps you follow to customize a workflow process. Specifically, you will:

1. Create a custom Identity Manager email template
2. Customize the Identity Manager Create User workflow process to use the new template and to send an email welcoming the new user to the company

NOTE

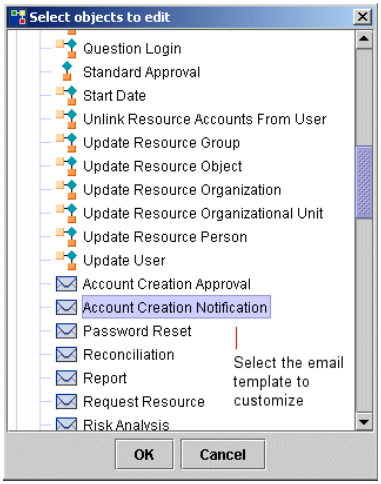
- Illustrations provided in this example may differ slightly from those you see when you load a process. Discrepancies can result in different component positioning or the selective removal of process actions that are unimportant to the example.
- Though you can perform many tasks from the tree view or diagram view, this example primarily uses BPE's tree view.

Step 1: Create a Custom Email Template

To create the custom email template, open and modify an existing Identity Manager email template as follows:

1. From the BPE menu bar, select File > Open Repository Object > Email Templates.
2. When the selection dialog displays (Figure A-45), select the Account Creation Notification template and then click OK.

Figure A-45 Selecting an Email Template



3. When the selected email template displays in the BPE, right-click the template name and select Copy from the pop-up menu.
4. Right-click again and select Paste.

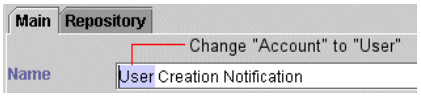
   A copy of the email template appears in the list view.

TIP	When pasting, be sure the mouse does not cover an item and that no items are selected (or the paste action will be ignored).

5. Double-click the new email template in list view to open the template.

6. Change the template name by typing **`User Creation Notification`** in the Name field.

Figure A-46 Renaming the New Template



7. In the newly created User Creation Notification template, modify the Subject and Body fields as needed.

Figure A-47 Customizing the User Creation Notification Email Template

You can also add a comma-separated list of Identity Manager accounts or email addresses to the Cc field.

8. When you are finished, click OK.

9. To save the template and check it into the repository, select File > Save in the Repository from the menu bar.

Now you are ready to modify the Create User workflow process. Continue to the next section for instructions.

Step 2: Customize the Workflow Process

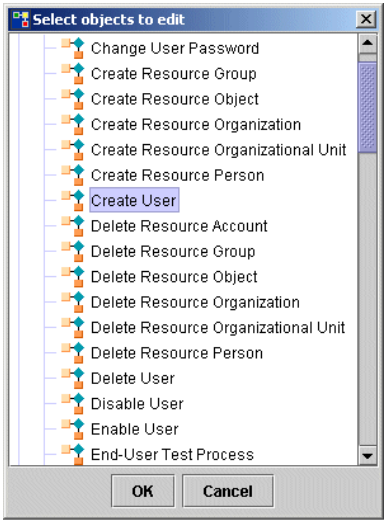
Use the following steps to modify the Create User workflow process to use the new email template:

1. Load the workflow process from the BPE by selecting File > Open Repository Object > Workflow Processes.

   A dialog appears that contains the Identity Manager objects you can edit.

2. Select the Create User workflow process, and then click **OK**.

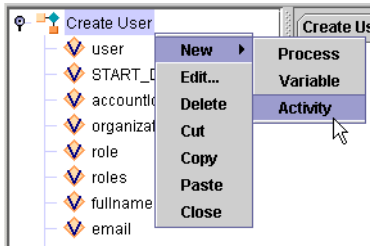
Figure A-48 Loading the Workflow Process



The Create User workflow displays.

3. In tree view, right-click the Create User process and select New > Activity from the pop-up menu.

Figure A-49 Creating and Naming an Activity



A new activity, named `activity1`, displays at the bottom of the activities list in the tree view.

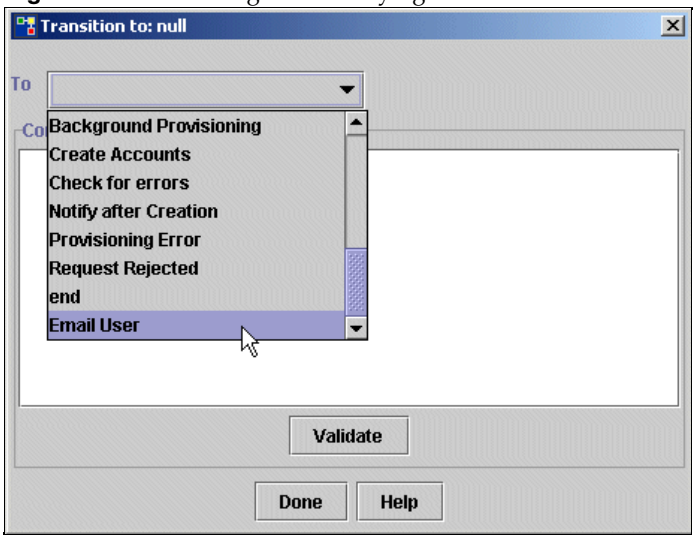
4. Double-click `activity1` to open the activity dialog.
5. Type **`Email User`** in the Name field to change the activity name.

   In the default Create User workflow, the step that notifies account requestors that an account was created (Notify) transitions directly to end.

   To include a new step in the workflow, you must delete this transition and create new transitions (between Notify and Email User, and from Email User to end) to send email to the new user before the process ends.
6. Right-click Notify, and then select Edit.
7. In the Activity dialog Transitions area, select end and then click Delete to delete that transition.
8. In the Transitions area, click New to add a transition.

9. When the Transition dialog displays, select Email User from the list and then click Done.

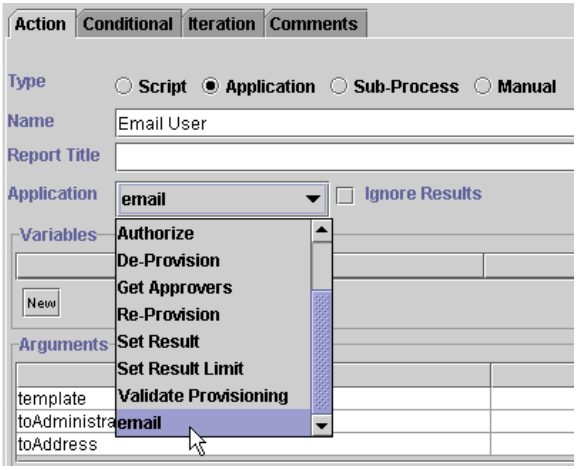
Figure A-50 Creating and Modifying Transitions



10. In the BPE tree view, right-click Email User, and then select New > Transition to create a transition and open the Transition dialog.
11. Select end, and then click OK.
12. Next, you must create an action for the new Email User activity that defines the email action and its recipient. In the tree view, right-click Email User, and then select New > Action to open the Action dialog.
13. Enable the Application button for the Type option.
14. Type a name for the new action in the Name field.

15. Select **email** from the Application menu.

Figure A-51 Creating an Action

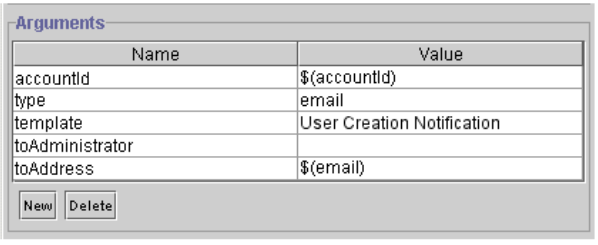


16. New selections display in the Arguments table. Enter the following information:

    - **template**: Enter the new email template name, **`User Creation Notification`**.
    - **toAddress**: Enter the **`$(user.waveset.email)`** variable for the user.

17. Click **New** to add an argument to the table. Name the argument **`accountId`** and enter the **`$(accountId)`** value for this argument.

Figure A-52 Creating an Action



18. When you are finished, click OK.

19. Select File > Save in the Repository from the BPE menu bar to save the process and check it back into the repository.

After saving, you can use Identity Manager to test the new process by creating a user. For simplicity and speed, do not select approvers or resources for the new user. Use your own email address (or one that you can access) so that, upon creation, you can verify receipt of the new welcome message.

Debugging Workflows, Forms, and Rules

The BPE includes a graphical debugger for workflows, rules, and forms. You can use the BPE debugger to set breakpoints visually, execute a workflow or form to a breakpoint, then stop process execution and examine the variables.

If you have previously used a code debugger for a procedural programming language, you will be familiar with the terms used in this section.

For more information about views, workflow, forms see the relevant chapters in *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

This section describes how to use the BPE Debugger, and is organized into the following sections:

Recommendations for Use

Use the BPE debugger under the following conditions only:

- **Use in a development or test environment**. Do not use the debugger in a production environment. Because setting a breakpoint is a global setting, incoming request threads are suspended when that breakpoint is reached.
- **Assign user the Run Debugger right**. (This right is granted as part of the Waveset Administrator capability.) The debugger can suspend threads (thereby potentially locking other users out of the system) and display variables, which possibly contain sensitive data, of other users' sessions. Given the powerful repercussions of misusing this right, exercise caution when assigning it.
- **Assign user a private copy of the application server**. If two users are developing on the same application server and one user connects a debugger to it, the other user will hit their breakpoints during usage, and will be locked out.

Clusters are not supported for use with the BPE debugger.

Running the Debugger Outside a Test Environment

If you find a problem in production that requires debugging, reproduce and debug it in a test environment. Setting breakpoint in the debugger can quickly bring down the application server in a production environment, where a large volume of traffic occurs. In addition depending on where breakpoints are set, users can be blocked from using the system.

If you cannot debug in a separate test environment, follow this procedure:

1. Divert all live traffic to a subset of your cluster by taking one of the nodes in your cluster offline. (For the purpose of this task, call this node server-a.)
2. Use the BPE to edit the System Configuration object by setting the `SystemConfiguration serverSettings.server-a.debugger.enabled` property to `true`.

   See "Step Two: Edit the System Configuration Object" on page 361 for more information about accessing the System Configuration object with the BPE.
3. Restart server-a so that the change to the System Configuration property setting can take effect.
4. Launch the debugger by selecting Tools > Debugger.

5. Create a new workspace, where the debugger connection uses the following URL:

   `server-a:<port>`

   When you have finished debugging

6. Set `serverSettings.server-a.debugger.enabled` to `false` and restart server-a to prevent the debugger from connecting to your live production environment.

7. Reintegrate server-a into your on-line cluster.

Disabling the Debugger

Always disable the `serverSettings.server-a.debugger.enabled` property in production to prevent someone from accidently connecting a debugger to the application server.

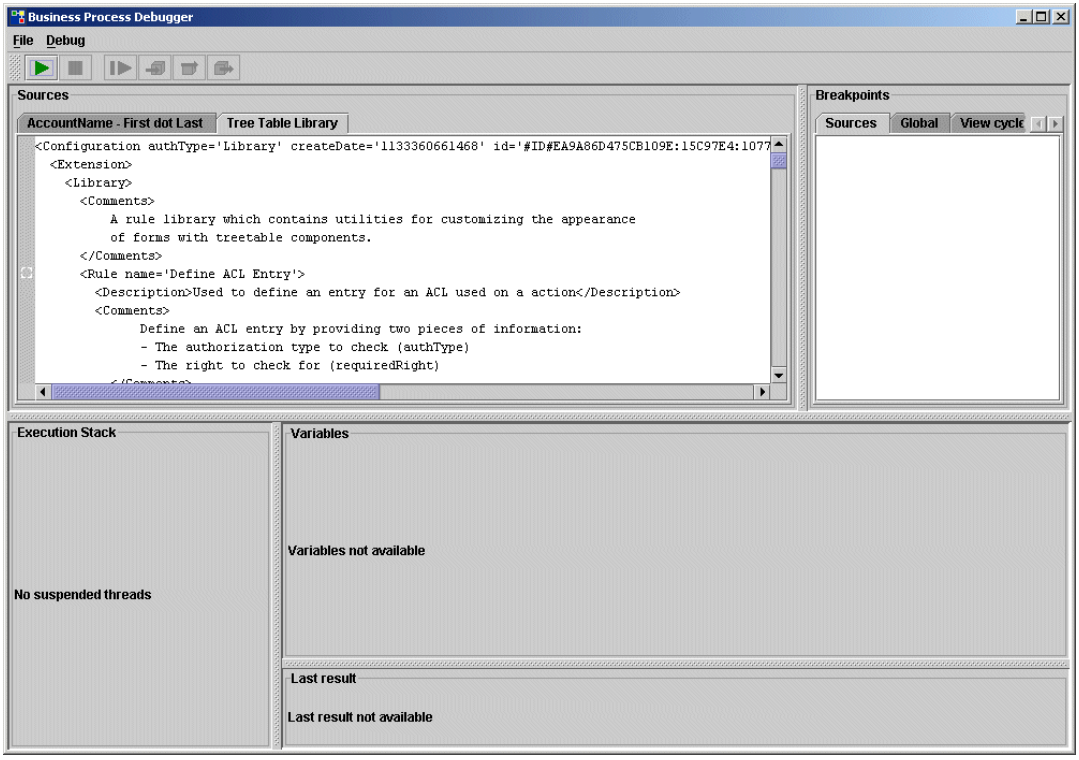
To disable the debugger, set the System Configuration object property `serverSettings.<server>.debugger`. to `enabled=false`.

Using the Debugger Main Window

The main debugger window displays the XML of the selected object and provides information about its execution. From this window, you can

- Start and stop the debugging process
- Navigate through the process execution
- Set distinct stopping points in process execution, or breakpoints. For more information about breakpoints, see Setting Breakpoints.

Figure A-53 BPE Debugger: Main Window



NOTE The BPE debugger provides numerous keyboard shortcuts for performing tasks. See "Using Keyboard Shortcuts" on page 311 for a list of these shortcuts.

The main window contains the following areas, which are described below:

- Sources Area
- Execution Stack
- Variables Area
- Variables Not Available
- Last Result

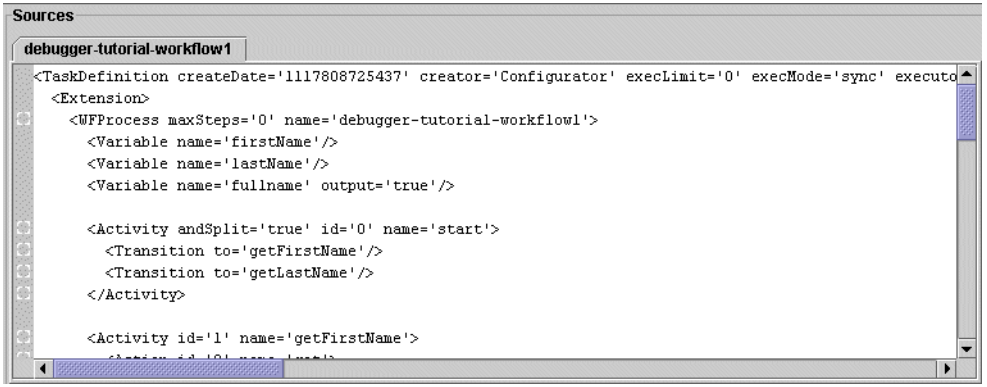
- Last Result Not Available
- Setting Breakpoints

Sources Area

The Sources area displays the unfiltered XML of the selected object.

The left margin of the XML panel displays a series of boxes that indicate points in the code where breakpoints can be set. Click the box immediately adjacent to the `<WFProcess...>` tag to set a breakpoint at the start of the workflow.

Figure A-54 BPE Debugger Main Window Source Panel

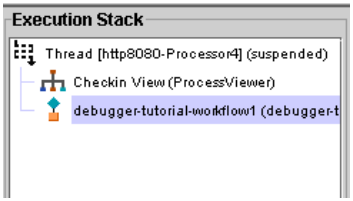


Execution Stack

The Execution Stack identifies which function in the selected object is under execution. This area lists the executing function's name and the name of the function that called it.

If additional functions appear in the call chain, these functions are listed in order. This list is also called a *stack trace*, and displays the structure of the execution stack at this point in the program's life.

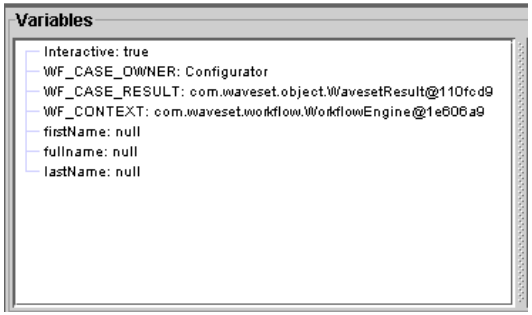
Figure A-55 BPE Debugger Main Window Execution Stack Panel



Variables Area

The Variables area lists all variables that are currently in scope at the current point of execution. Click on the variable object name to expand it and display the names of each variable.

Figure A-56 BPE Main Window Variables Panel



Variables Not Available

The Variables Not Available area displays if debugging is inactive or if the selected stack frame is not the current stack frame.

Last Result

If the current element is an XPRESS end tag, the Last Result area contains the result of that evaluation. Also applies to other tags for which last value makes sense. For example, as `<Argument>`'s to workflow subprocesses are evaluated, this area has the value of that argument. This area is not available if debugging is not currently in progress.

Figure A-57 BPE Debugger Main Window Last result Panel

Last Result Not Available

The Last Result Not Available area displays if debugging is inactive.

Setting Breakpoints

A *breakpoint* is a command that the debugger uses to halt the execution of the object before executing a specific line of code. In the Identity Manager debugger, code breakpoints apply regardless of where the form or workflow is launched from.

While most debuggers allow you to set breakpoints only on source locations, the BPE debugger permits you to also set breakpoints at conceptual execution points, such as Refresh view. In this case, the debugger will suspend when a Refresh view operation occurs. You can then step-into the refresh view and see the underlying form processing in progress.

Setting a breakpoint is a global setting. That is, it causes the incoming request threads to suspend when the designated breakpoint is reached. This happens regardless of which user is making the request.

Setting a Breakpoint

To view a summary of all source breakpoints, click the Sources tab. The Breakpoints pane lists all source breakpoints. Navigate to a particular breakpoint by clicking the breakpoint in.

Types of Breakpoints

The Breakpoints area provides the following types of breakpoint settings:

- **Global breakpoints** (Global tab)
- **Breakpoints associated with commonly used views** (View cycle tab)
- **Breakpoints associated with stages of form processing** (Form cycle tab)

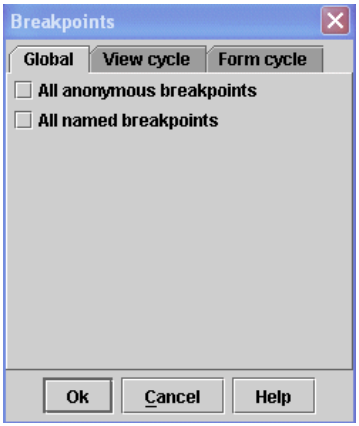
Access these breakpoint types by selecting the designated tab.

- Select the Global tab to set code breakpoints by:
  - **All anonymous breakpoints**: Sets a breakpoint on an anonymous source.
  - **All named breakpoints**: Turns step-over and step-out into step-into processing.

    Breakpoints always supersede step-over and step-out functionality. Consequently, if you enable this setting, you have effectively turned step-over and step-out into step-into processing. In typical use, set All named breakpoints only if you do not know which form or workflow a given page uses. If you turn on this setting, turn if off immediately after the debug process identifies the form or workflow. Otherwise, you will be forced to step through every point of execution.
  - Enabling both settings, results in the debugger checking all breakpoints.

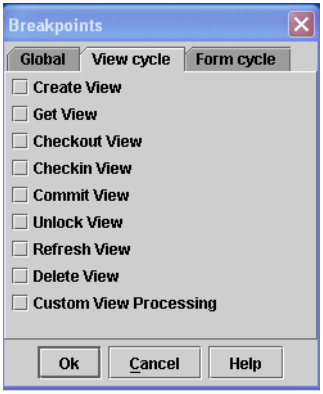
(Optional) Select a global setting, and click OK.

Figure A-58 BPE Debugger Breakpoints Panel: Global Tab



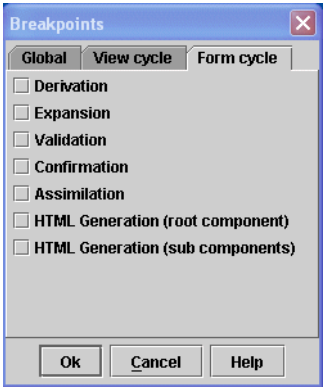
- Select the View cycle tab to set code breakpoints based on the view processing that occurs during process execution. The most commonly invoked view operations are listed in this dialog. Each of the listed view operations are available on each view.

Figure A-59 BPE Debugger Breakpoints Panel: View Cycle Tab



- Select the Form cycle tab to set code breakpoints based on a designated stage of form processing. For information about the stages of form processing, see *Sun Java™ System Identity Manager Workflows, Forms, and Views*.

Figure A-60 BPE Debugger Breakpoints Panel: Form Cycle Tab



Stepping through an Executing Process

Stepping through describes the sequential, deliberate analysis of an executing process' functions.

Terminology

Step into, *step over*, and *step out* are terms borrowed from debuggers of procedural programming languages, in which the execution order is implied by the structure of the language. However, in Identity Manager forms and workflow, the order in which elements occur in the code do not reflect the order in which they are executed.

Consequently, these terms have slightly different meanings when used in the Business Process Editor:

- **Step-into**: Describes moving to the next point of execution on the current thread. *Step-into* always the smallest amount by which you can proceed through a process in the debugger XML display.
- **Step over**: Describes moving from the current `begin` tag to the current `end` tag without stopping at an interim element. Stepping over permits you skip almost everything between a `start` and `end` tag. However, if the next point of execution does not occur between the `start` and `end` tags of the current element, debugging halts there instead.

For example, in a workflow containing multiple active virtual threads, you can step to the `start` tag of an action, but the next element to be executed is a different action. In this case, the process stops executing at a different point. Thus, you avoid accidently skipping over a potentially significant element.

- **Step out**: Describes moving incrementally until the execution stack is one less than the current. Similar to step-over. If the next point of execution has a different parent execution stack, it will stop there instead.

General Hints

Following is a list of hints to help you successfully step through an executing process:

- Set up stepping in the debugger to be as granular as feasible in the context of your debugging task. This practice helps you avoid missing anything potentially critical to debugging.
- Stepping does not change the execution order of your program. The program's execution order is the same as it would be if the debugger were not attached. You can skip seeing portions of the execution (but they still execute regardless).
- Click step-into when you want the smallest step possible through the code.
- Click step-over when you feel that no probable problem exists with the content between the start and end tag. The debugger then skips this element although the code in between these tags still executes.

Table A-7 provides a snapshot of how the BPE debugger would proceed through the following code sample:

```
<A>
   <B/>
</A>
<D/>
(A, B, and D are some xml elements)
```

Table A-7 Example Debugging Process

Execution Order	Result
<A>, <B/>, </A>, <D/>	If you are clicking **step-into**, the debugger highlights the lines in that execution order.
	If you are clicking **step-over**, the debugger highlights <A>, </A> (skipping B), <D/>
<A>, <D/>, <B/>, </A>	If you are clicking **step-over**, you will see code lines in the order <A>, <D/>, <B/>, </A>. (Step-over is equivalent to step-into for this case.)

Getting Started

The BPE includes a tutorial on using the debugger with workflow, forms, and rules. The debugger ships with `sample/debugger-tutorial.xml`, which contains some sample workflows, rules, and forms. These samples are used throughout this chapter for tutorial purposes.

Step One: Import Tutorial File

Use one of the following methods to import the tutorial file:

- From Identity Manager, select Configure > Import Exchange File. Then either enter **`sample/debugger-tutorial.xml`** into the File to Upload field or click Browse to navigate to this file.
- From the Console, enter **`import -v sample/debugger-tutorial.xml`**.

After importing the file successfully, continue to the next section.

Step Two: Edit the System Configuration Object

To edit the system configuration object, use the following steps:

1. From the BPE, open the System Configuration object for editing by selecting File > Open Repository Object > Generic Objects > System Configuration.
2. In the tree view, expand `serverSettings` and the `default` attribute, and then select `debugger`.
3. In the Attributes panel, click the Value column to enable debugging.
4. Select File > Save In Repository to save your change.
5. Restart your application server.

CAUTION *Do not* enable this property in production.

Step Three: Launch the Debugger

After restarting the application server, you can select Tools > Debugger to launch the BPE debugger.

Example: Debugging the Tabbed User Form and Refresh View

This section provides a sample debugging procedure to illustrate how debugger breakpoints apply regardless of from where you launched a form or workflow.

The sample procedure includes the following steps:

1. Setting a Breakpoint
2. Creating New User
3. Viewing Before Refresh View Results
4. Viewing After Refresh View Results
5. Stepping Through the Form
6. Completing the Form Processing

Setting a Breakpoint

To set a breakpoint:

1. Click the View cycle tab in the Breakpoints panel.
2. Check Refresh View. The debugger now executes a breakpoint whenever a view is refreshed during execution.

Creating New User

To create a new user:

1. In Identity Manager, select Accounts > New... User.
2. Enter a first name (for example, **`jean`**) and a last name (for example, **`faux`**).
3. Click the Identity tab to trigger a refresh view operation.

Viewing Before Refresh View Results

Return to the debugger frame, which is now suspended on the Refresh view breakpoint that you set. The Execution Stack lists Before Refresh View, which indicates the state of the view just before the refresh operation occurred. The Variables panel displays the view just before it has been refreshed.

Figure A-61 Example 1: Debugging Suspended on Before Refresh View Breakpoint

Expand the global subtree and locate the `firstname` and `lastname` values that you typed in the form. Note that the `fullname` is currently `null`.

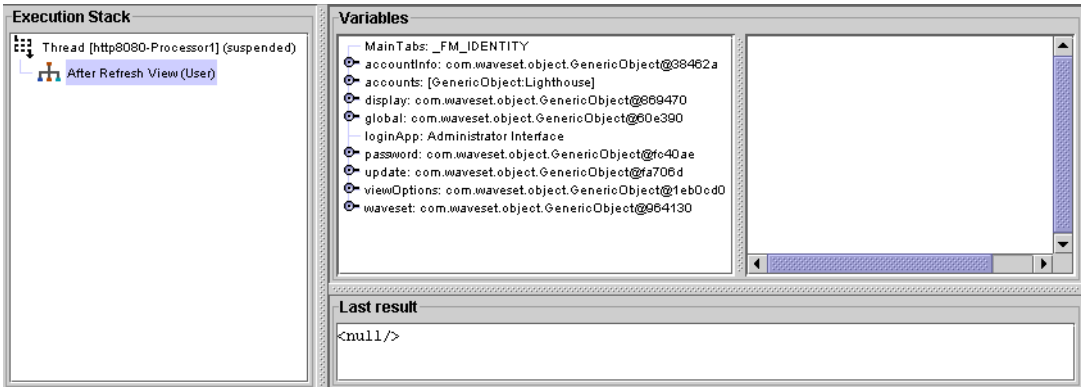
Viewing After Refresh View Results

To view the After Refresh view results,

1. Click Continue.

   The Execution Stack lists After Refresh View, which indicates that it now displays the state of the view just after refresh has occurred. Note that the `fullname` value is now **`jean faux`**.

Figure A-62 Example 1: Debugging Suspended on After Refresh View Breakpoint



2. Click Continue again.

   The form has resumed execution. Return to the browser window. Change First Name to **`jean2`** and click the Identity tab again to trigger another refresh.

3. Return to the debugger frame.

   Form processing is suspended at Before Refresh View.

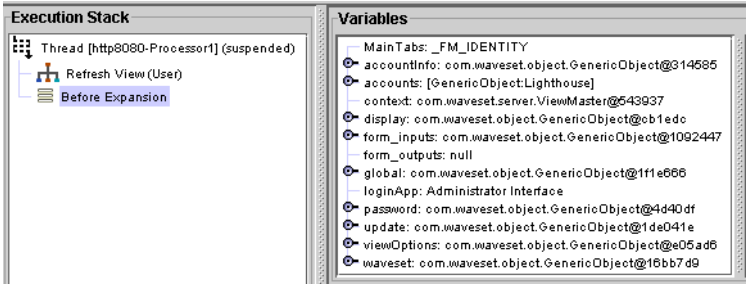
Stepping Through the Form

To Step Through the form,

1. Click Step-Into to reveal the fullname expansion in execution.

   The debugger displays Before Expansion, which indicates that the form variables have not been expanded.

Figure A-63 Example 1: Debugging Suspended Before First Expansion Pass



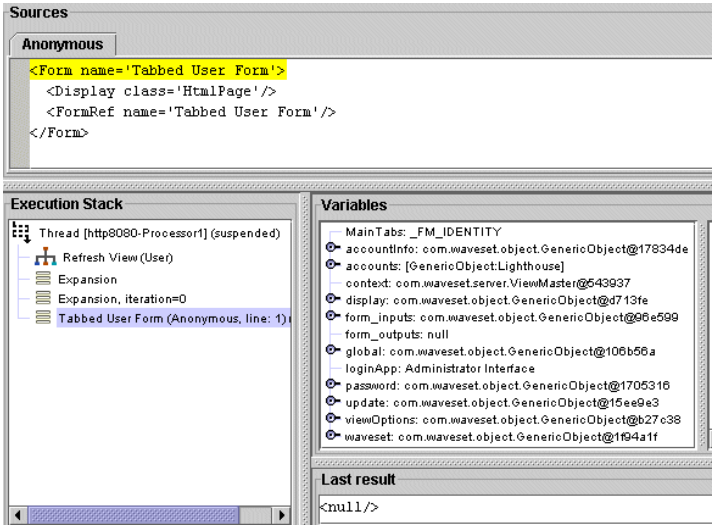
2. Click Step-Into again.

   The debugger displays `Before Expansion, iteration=0`, indicating that you will see the form variables before the first Expansion pass.

3. Click Step-Into again.

   The debugger is now on an anonymous source. The anonymous source is a wrapper form created on the fly and is related to the MissingFields form.

Figure A-64 Example 1: Stepping-into the Start of Tabbed User Form



4. Click Step-Into two more times until you reach the beginning of Tabbed User Form.

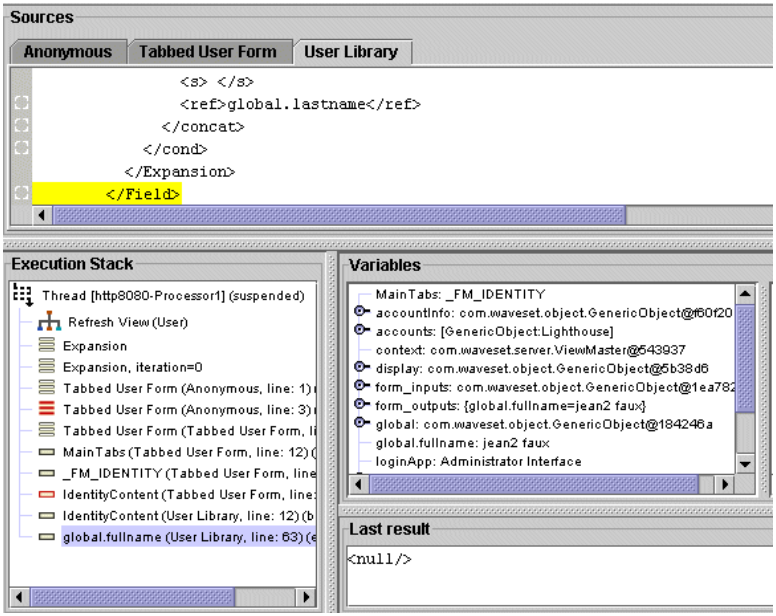
5. Continue to click Step-Into until you reach `<Field name='global.fullName'>` (Approximately 20 to 30 step-into operations.)

6. Click Step-into 15 times or until you have reached the `</Field>` element.

   While stepping, the Last result at the `</concat>` tag is **`jean2 faux`**.

   The `form_outputs` contains `global.fullname: jean2 faux`.

Figure A-65 Example 1: Completed Debugging of Tabbed User Form



Complete Form Processing

To complete form processing:

1. Click Step-out seven times.

   At this point, your stack should indicate:

   ```
   Refresh View (User)
   After Expansion
   ```

   The Variables panel reflects the state of the form variables after all expansions have run.

2. Click Step-out again.

   You have now reached After refresh view. The variables now displayed are the view variables.

3. Expand the global subtree.

   Note that `fullname` is now **`jean2 faux`**.

4. Click Continue.

Debugging Workflows

This section provides information about debugging your workflows.

The Workflow Execution Model

Workflows are executed by a single Java thread and are represented in the Execution Stack panel by a single Java thread. However, within a workflow, each activity becomes its own virtual thread.

During workflow execution, the workflow engine cycles through a queue of virtual threads. Each virtual thread is in one of the states described in the following table.

Table A-8 Virtual Thread States

Workflow Activity State	Definition
ready	Identifies an activity that has just been transitioned to. (This state is very temporary, as actions typically start executing immediately after being designated ready.)
executing	Identifies an activity that contains one or more actions that are currently being executed or have yet to run. This is a logical state, which does not mean that the Java thread is currently executing it. The action currently being executed is always the action that is highlighted in the debugger.
pending outbound	Identifies an activity after all actions within an activity have been run, it goes to the pending outbound state. In this state, it awaits an outbound transition to be taken. In the case of an or-split, it is in this state until one transition is taken. In the case of an and-split, it will be in this state until all transitions whose conditions evaluate to true are taken.
inactive	Identifies an activity in which all transitions have been taken.
pending inbound	Identifies a virtual thread whose activity is an and-join. That is, one transition to this virtual thread has occurred, but the process is still waiting for other transitions.

After all transitions have completed, the workflow process subsequently begins executing.

Example 1: Debugging a Workflow and a Rule

The example provided in this section illustrates how to use the BPE debugger to debug a sample workflow and rule using a workflow provided in `debugger-tutorial-workflow1` (supplied with Identity Manager). This example exemplifies how to step-into and step-through workflow debugging and rule execution.

For this example, you perform the following steps:

1. Launch the process.
2. Start execution.
3. Step through the `getFirstName` thread.
4. Step into and over the `getlastname` thread.
5. Step into `computefullname` processing.
6. Step through rule processing.
7. Conclude workflow processing.

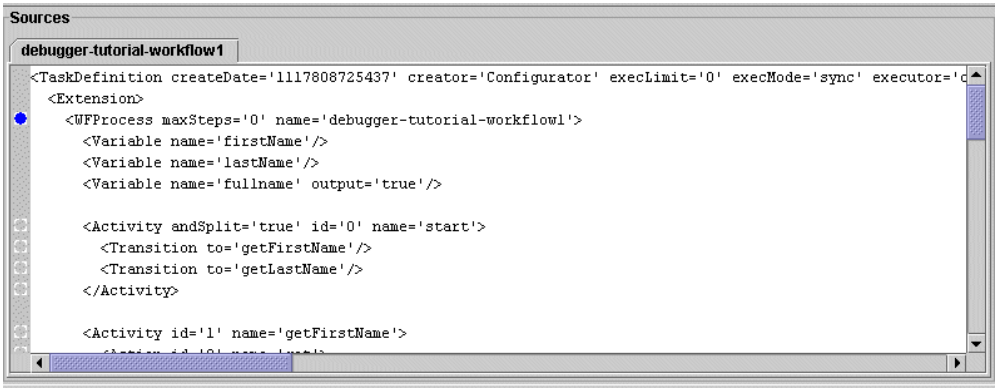
Step One: Launch the Process

To launch the workflow debugging process:

1. From the debugger main window, select File > Open Repository Object.
2. Click debugger-tutorial-workflow1.

   Note the small boxes in the left margin of the XML display. These boxes identify potential breakpoints that you can insert into the code.

Figure A-66 Setting the First Breakpoint

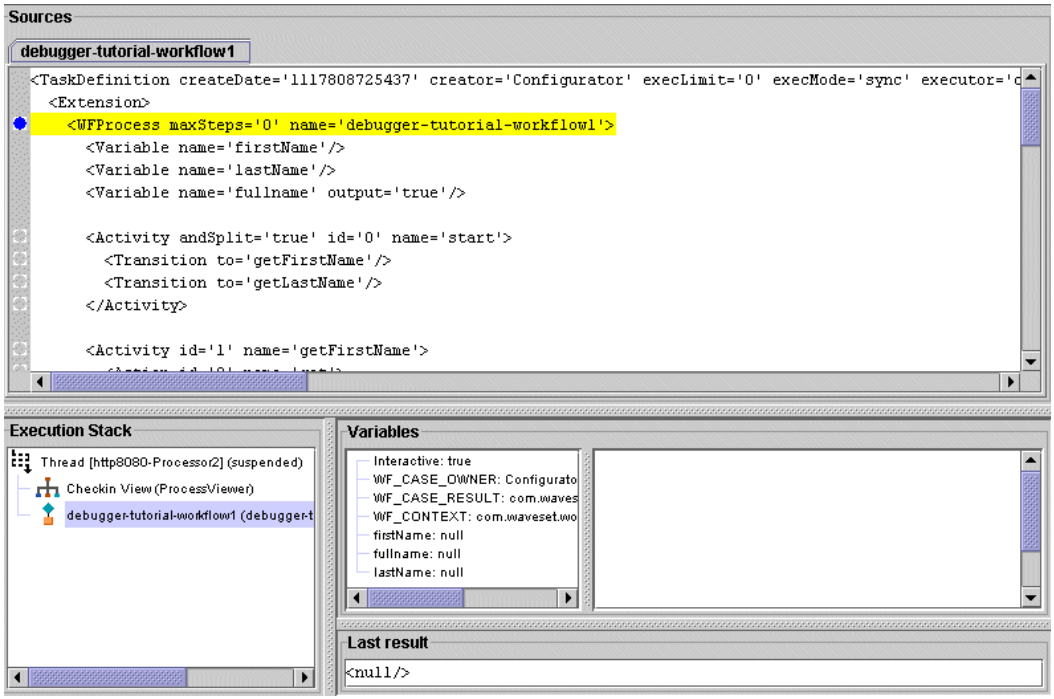


3. Click the box adjacent to the `<WFProcess>` tag to set a breakpoint at the start of the workflow.
4. Log in to Identity Manager and select Tasks > Run Tasks.

5. Click debugger-tutorial-workflow1.

   The debugger frame indicates that debugging has halted at your breakpoint.

Figure A-67 Debugging Halted at Breakpoint



Note the following:

- **Execution Stack Panel** — At the top of the Execution Stack panel, you see `Thread [thread name] (suspended)`, which indicates that this workflow is currently being run by the thread of the given name, and that it is suspended at the breakpoint you set.

  Below the Thread is your execution stack. This stack is an upside-down stack trace, with the calling function on top and the called function at the bottom. (It is upside down compared with how most debuggers represent execution stacks.)

The top most frame in the stack says Checkin View (ProcessViewer), which indicates that the workflow is being called by the `checkinView` method of the ProcessViewer. Because you do not have access to the Java source code for this stack frame, clicking on it will not display new information. However, the stack frame does provide context about where the workflow is being launched from.

The next frame in the stack is highlighted because it corresponds to the current point of execution, which is the beginning of the workflow process (`<WFProcess>`).

- **Variables panel** — Lists all variables that are currently in scope at the current point of execution. You will see
  - **Interactive** — This variable is passed in by the view as an input to the process.
  - **WF_CASE_OWNER, WF_CASE_RESULT, WF_CONTEXT** — These variables are implicit workflow variables.
  - **firstName, fullname, and lastName** — These variables are declared in the workflow using `<Variable>` declarations.

6. Select Debug > Select Current Line (F5) to re-highlight your current line of execution.

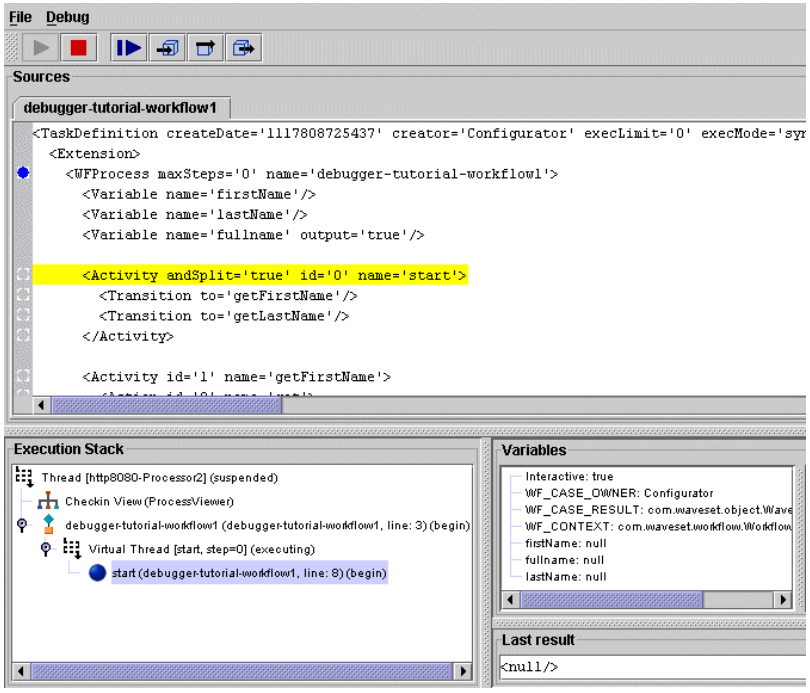
Step Two: Start Execution

To start execution:

1. Click Step-Into.

   At this point, the debugger moves to the start activity. Notice that the execution stack contains a Virtual Thread `[start, step=0]` (executing), which indicates there is a Virtual Thread for the start activity that is currently in the executing state.

Figure A-68 Stepping-into the Execution of the First Virtual Thread



2. Click two levels up on the `debugger-tutorial-workflow-1` frame to highlight the `WFProcess`, showing you the location of the caller.
3. Press F5 to return to the current line.
4. Click Step-Into.

   At this point, the debugger moves to the `</Activity>` and the start virtual thread is now pending outbound.

Step Three: Step Through the getFirstName Thread

Use the following procedure to step-through the `getFirstName` thread:

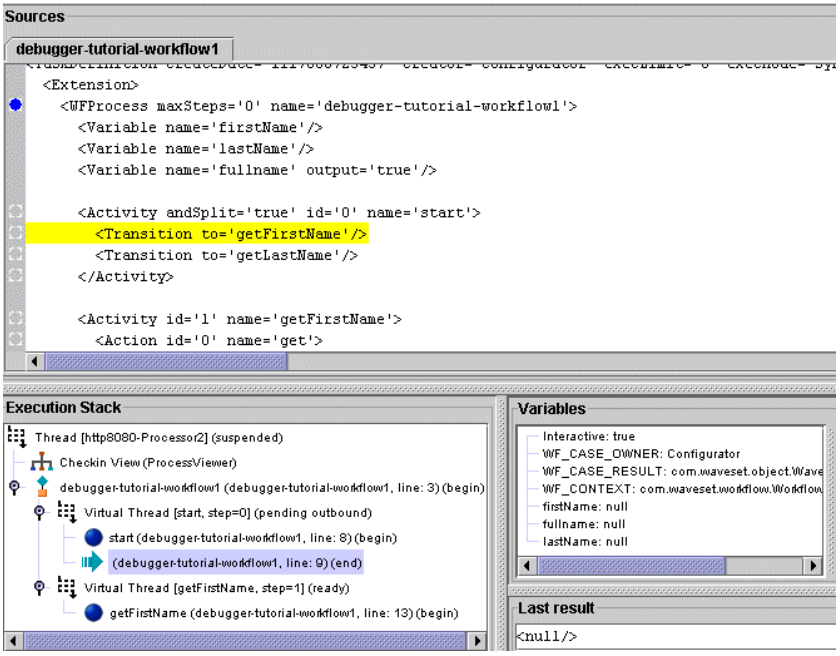
1. Click Step-Into.

   At this point, the debugger has highlighted the transition to `getFirstName`.

2. Click Step-Into.

   A new Virtual Thread for `getFirstName` has been created as a result of this transition. This Virtual Thread is currently in the ready state. The start virtual thread is still pending outbound (because this is an and-split operation, it must take all possible transitions).

Figure A-69 Example 2: Stepping-into the Execution of getFirstName



3. Click Step-Into again.

   The debugger jumps to the `getFirstName` activity. The state changes from ready to executing.

4. Click Step-Into.

   The debugger moves to the `get` action.

5. Click Step-Into three more times or until the debugger reaches the `</set>` tag.

   The variables panel indicates that `firstName` has been set to `myfirstname` as a result of the `</set>`.

Step Four: Step Into and Over the getLastName Thread

Use the following procedure to step-into and over the getLastName thread:

1. Click Step-Into three more times or until the debugger reaches the `</Activity>` for `getFirstName`.

   The `getFirstName` virtual thread is now pending outbound.

2. Click Step-Into.

   The debugger returns to the start virtual thread, and is about to process the transition to `getLastName`.

3. Click Step-Into.

   The start has gone to inactive because all transitions have been processed. `getLastName` is now in the ready state because of this transition.

4. Click Step-Into.

   At this point, the start virtual thread will go away because it is inactive. Debugging moves to the `getLastName` virtual thread, which is now in the executing state.

5. Click Step-Over to skip to the end of `getLastName`.

   The `lastName` variable in the variables panel has been set to `mylastname`. Both the `getFirstName` and `getLastName` virtual threads are pending outbound.

6. Click Step-Into.

   The debugger is on the transition from `getFirstName` to `computeFullName`.

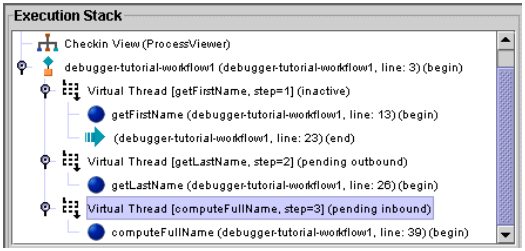
7. Click Step-Into.

   `getFirstName` goes to inactive and a new virtual thread, `computeFullName` is created. This thread is in the pending inbound state because it is still waiting on the inbound transition from `getLastName`. (The wait occurs because it is an `and-join` operation. If it were an `or-join` operation, process status would immediately go to ready.)

8. Click Step-Into.

The debugger is now on the transition from `getLastName` to `computeFullName`.

Figure A-70 Debugger Transitioning from getFirstName to computeFullName



Step Five: Step Into computeFullName Processing

Use the following procedure to step-into `computeFullName` processing:

1. Click Step-Into.

   The `computeFullName` virtual thread goes from pending inbound to ready because of this transition.

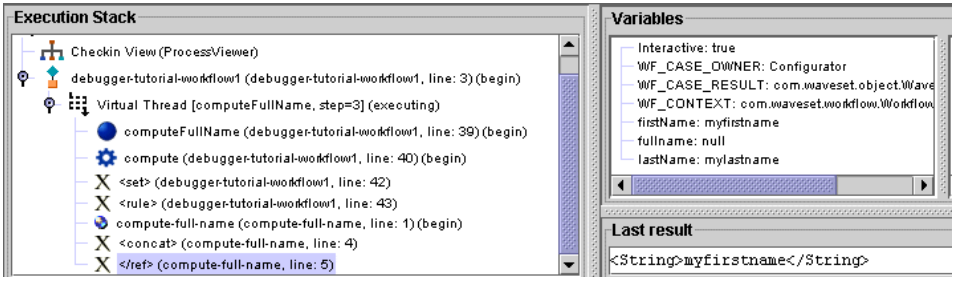
2. Click Step-Into.

   `computeFullName` is now executing.

3. Click Step-Into five more times.

   The debugger is now on the `</argument>` tag for `firstName`. The last result panel shows `<String>myfirstname</String>`. This value is passed for the `firstName` argument.

Figure A-71 Stepping Into computeFullName Processing



Step Six: Step Through Rule Processing

To Step Through rule processing:

1. Click Step-Into three more times.

   The debugger steps into the Compute-Full-Name rule. In the execution stack, click the frame to move up one frame. The `<rule>` call in debugger-tutorial-workflow-1 is highlighted to indicate from where the rule is being called. Press F5 to re-select your current line.

2. Click Step-Into three more times or until the debugger reaches the `</ref>` tag.

   The last result panel shows `<String>myfirstname</String>`, which is the result of the `<ref>firstName</ref>`.

3. Click Step-Into three more times or until the debugger reaches the `</concat>` tag.

   The Last result panel displays the result of the `<concat>` expression.

   ```
   <String>myfirstname mylastname</String>
   ```

4. Click Step-Into twice more and Debugging returns to the `</rule>` tag.

Step Seven: Concluding Workflow Processing

To conclude workflow processing:

5. Click Step-Into until you reach the `</set>` element.

   The `fullname` variable has been updated to `myfirstname mylastname`.

6. Click Step-Into twice more.

   `computeFullName` is now pending outbound.

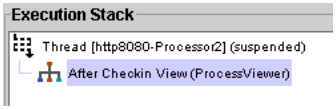
7. Click Step-Into four more times. `end` goes to ready, then executing.

   The debugger reaches the `</WFProcess>` tag, indicating that the process has now completed.

8. Click Step-Into.

   The Execution Stack displays After Checkin view, meaning that the checkin-view operation that launched this workflow has completed.

Figure A-72 Example 2: Completion of Check-in View Operation



9. Click Continue to resume execution.

   If the browser request has not timed out, the Task Results diagram with the process diagram is displayed.

Example 2: Debugging a Workflow Containing a Manual Action and a Form

This section provides an example that illustrates how to debug a sample workflow containing a manual action and a form.

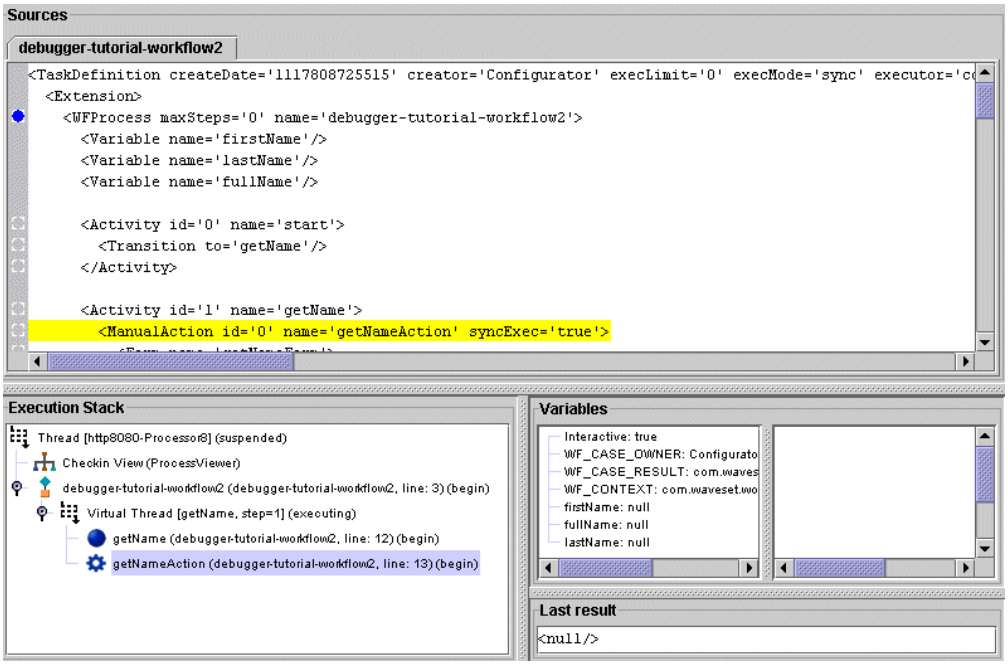
Use `workflow2` from the debugger tutorial files and perform the following steps:

1. Select File > Open Repository Object.
2. Expand Workflow Processes, and select `debugger-tutorial-workflow2`.
3. Set a breakpoint on the `<WFProcess...>` tag.
4. Log in to Identity Manager, and navigate to Tasks > Run Tasks.
5. Click `debugger-tutorial-workflow2`.

   The debugger has stopped at the breakpoint you set.

6. Click Step-Into six times or until the debugger marks `<ManualAction... name='getNameAction'>)`.

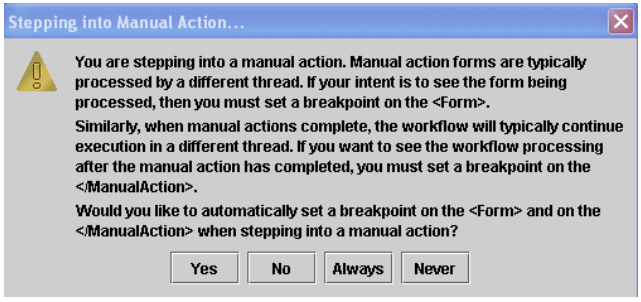
Figure A-73 Stepping Into a Manual Action



7. Click Step-Into.

8. When a dialog displays to explain that form processing occurs in a different thread, set a breakpoint on the `<Form>` tag to see the processing occur.

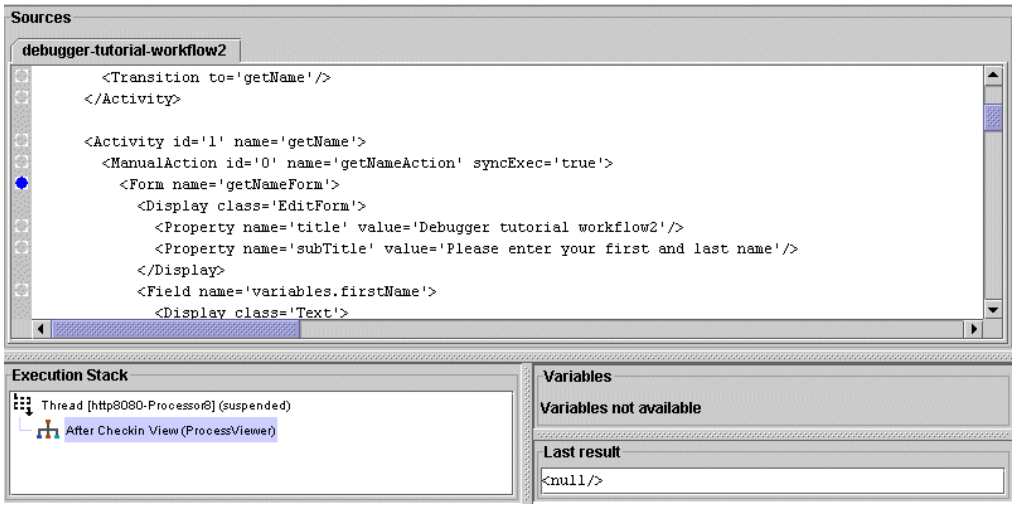
Figure A-74 Stepping Into Manual Action Dialog



9. Click Yes or Always.

   After form processing has completed, the workflow continues execution in a separate thread. Consequently, you must set a breakpoint on the `</ManualAction>` to observe workflow processing after the form has completed processing.

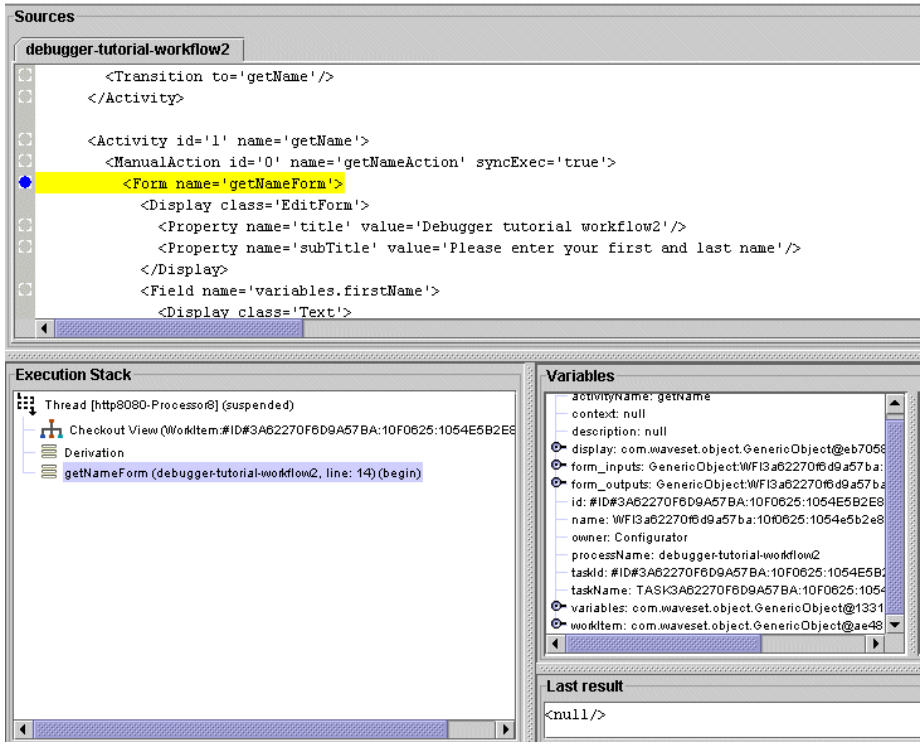
Figure A-75 Breakpoint Marking Start of Form



   Note that the debugger has set breakpoints on the `<Form>` and `</ManualAction>` tags as indicated. In addition, the execution stack indicates After checkin view... and you have stepped out of the workflow process because workflow processing has proceeded as far as possible (until the manual action completes).

10. Click Continue, and the debugger stops processing at the breakpoint set on the `<Form>` element.

Figure A-76 Debugger Displaying Manual Action Processing



Note the following In the Execution Stack area:

- **Checkout View (WorkItem:...)** — Indicates that processing is occurring in the context of a checkout view for the given work item.
- **ManualAction forms** — Operate against the work item view and manipulate workflow variables through the variables object. Expand the variables object to see the non-null workflow variables.
- **Derivation** — Indicates that form execution is on the Derivation pass.

11. Because this form contains no `<Derivation>` expressions, proceed to the next phase or processing by clicking Continue. The HTML Generation (root component) pass of form processing begins.

HTML Generation Phase (Root Component)

To generate HTML for the root component:

1. Click Step-Into twice.

   The debugger has just processed the title `<Property>` element. The Last result panel contains the value of this property.

2. Click Step-Into three more times.

   The debugger skips the fields in the form and goes directly to the `</Form>` element because this pass focuses only building the root component of the page.

3. Click Continue.

   The HTML Generation (subcomponents) pass of form processing begins.

HTML Generation (Subcomponents)

To generate HTML for the subcomponents:

1. Click Step-Into 13 times or until the debugger reaches the `</Form>` tag.

   The debugger iterates over each of these fields and evaluates their display properties.

2. Click Continue.

   The debugger displays No suspended threads because execution has resumed. Control has now returned to your browser window.

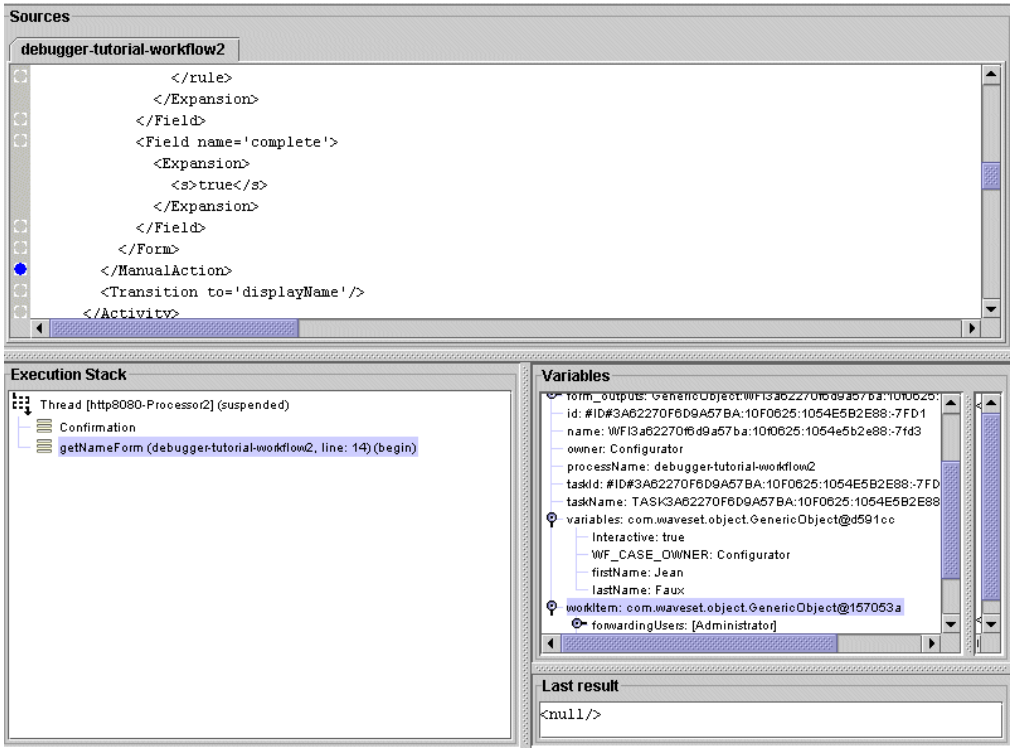
3. Return to your browser window, and enter your first and last name as prompted, and click Save.

   Return to your debugger frame. The debugger is now suspended on your breakpoint.

4. Expand the Variables subtree.

   The `firstName` and `lastName` are the values that you just entered. The debugger is currently in the Confirmation phase of form processing.

Figure A-77 Form Processing Confirmation Phase



Confirmation

Because this form has no confirmation fields, no processing occurs. Click Continue to begin the Validation phase of form processing.

Validation and Expansion

Because this form contains no Validation expressions, no obvious processing occurs.

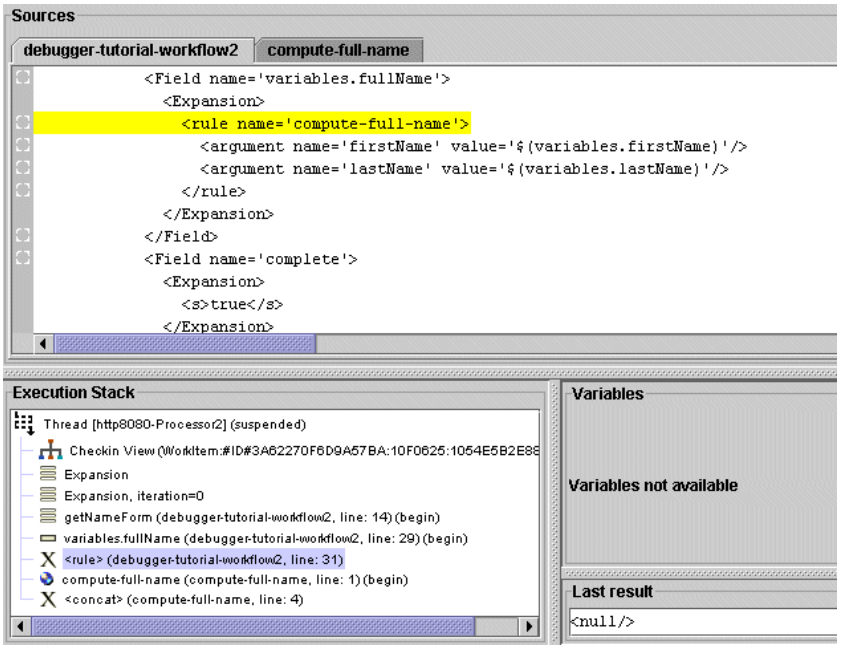
1. Click Continue to skip the Validation phase.

   You are now on the Expansion phase of form processing.

2. Click Step-Into six times.

   The debugger is now on the `<rule>` tag of the `<Expansion>` of the `variables.fullName` field.

Figure A-78 Stepping Into Rule Processing



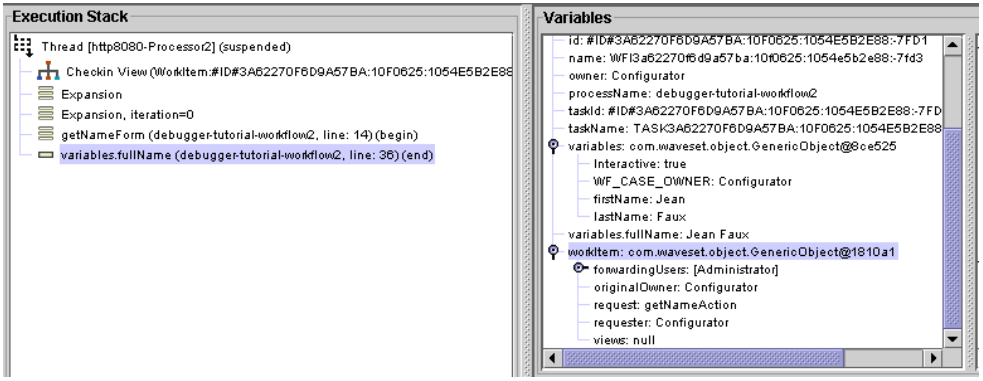
3. Click Step-Into five times, and the debugger has now stepped into the `<rule>` element.
4. Click Step-Into seven times or until the debugger reaches the `</Rule>` element.

   The Last result contains the full name.
5. Click Step-Into again and processing resumes in the form.
6. Click Step-Into again.

   The top-level `variables.fullName` has the value of the Expansion expression that just ran. This is top-level entity rather that a child of the `variables` data structure because during form processing, form outputs are kept in their own temporary `form_outputs` data structure, with path expressions flattened.

   After form processing, form outputs are assimilated back into the view. In the implicit variables `form_inputs` and `form_outputs`, `form_inputs` shows the unmodified workitem view, and `form_outputs` shows the output fields that are assimilated back into the view after form processing completes.

Figure A-79 Debugger Displaying Completed Execution of variable.fullName

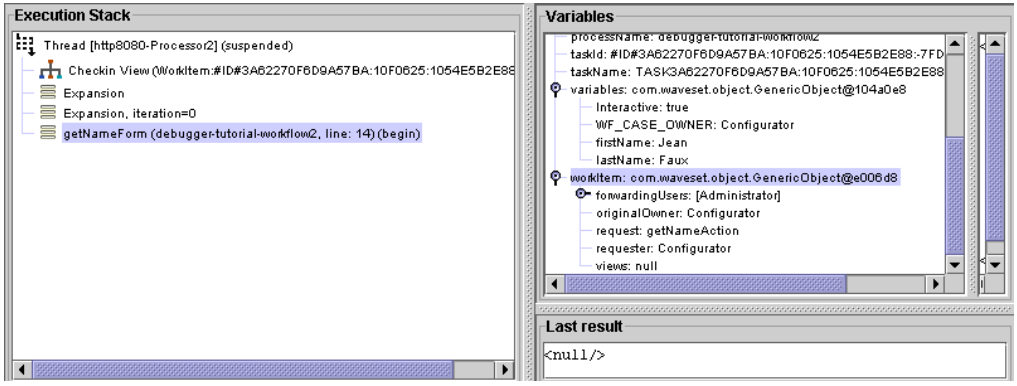


In general, `form_inputs` identifies the view, and `form_outputs` contains data to be assimilated back into the view. However, not all forms are necessarily tied to a view (for example, active sync forms). The form engine is a general data mapping engine, mapping from form inputs to form outputs. The view handler is responsible for passing the view into the form engine and assimilating the outputs back into the view.

7. Click Continue.

   The debugger reaches the `</ManualAction>` breakpoint, which was set before when the debugger stepped into the Manual Action. The variables `firstName` and `lastName` are the values that you entered. `fullName` is the result of the Expansion expression that was just run.

Figure A-80 Debugger Displaying the Result of Expansion Processing



8. Click Step-Into five times until `<ManualAction... name='displayNameAction'>`.
9. Click Step-Into again. (Click Yes or Always, if prompted.)
10. Click Continue.

    The debugger is now on the Derivation pass for `displayNameForm`.

Derivation and HTML Generation (Root Component)

To complete the derivation and HTML generate phase

1. Click Continue to begin the HTML Generation (root component) processing for `displayNameForm`.
2. Click Step-Into eight times or until the debugger reaches the `</Property>` element for `subTitle`.
3. Click Continue twice.

   The debugger displays the following message:

   ```
   No suspended threads because execution has resumed. Control has now
   returned to the browser window.
   ```

4. Return to your browser window.

   The information displayed is the same you entered.

5. Click Save and return to your debugger frame.

   The debugger is now on the Confirmation pass, processing the `displayNameForm`.

Validation and Expansion

To begin validation and expansion,

1. Click Continue to begin the Validation pass.
2. Click Continue to begin the Expansion pass.
3. Click Continue again.

   The debugger is now on the `</ManualAction>` tag because the manual action is complete. At this point, workflow processing has resumed.

4. Click Step-Into five times or until the debugger reaches the `</WFProcess>` tag, indicating that the workflow has completed execution.

5. Click Continue.

   The debugger displays the following message:

   ```
   No suspended threads because resumed execution has resumed. Control has
   now returned to your browser window.
   ```

6. Return to your browser window to observe the workflow process diagram.

Debugging Forms

Forms are processed in a series of passes. Depending on which pass is in progress, certain elements are processed and others are ignored. The execution stack in the debugger main window indicates the current phase of form processing. The execution stack frame preceding the outermost form always has the name of the pass.

Derivation

During Derivation phase of form execution, the form engine iterates over each field and it processes each `<Disable>` expression.

The form engine also processes the `<Derivation>` expression, for fields whose `<Disable>` expressions return `false`.

Expansion

The debugger iterates over each field and processes each `<Disable>` expression. For those fields whose `<Disable>` expressions return `false`, it processes the `<Expansion>` expression.

The Expansion processing phase continues to run until either

- No more changes occur.
- `maxIterations` has been exceeded. `maxIterations` is a parameter that is passed into the form engine form Expansion element processing.

By default, `maxIterations` is set to one. Consequently, the debugger makes only one pass, which is why the Execution stack panel displays `Expansion, iteration=0` when expansions are run.

Validation

The debugger iterates over each field, processing each <Disable> expression. For those fields whose <Disable> expressions return false, the form engine also processes the following:

- If the field has a <Display> expression with a required property, it evaluates that property expression
- If the field has a <Validation> expression, it evaluates that validation expression

Confirmation

The debugger iterates over each field, processing each <Disable> expression. For those fields whose <Disable> expressions return false and that also have a confirm attribute, the debugger confirms that the field referenced by confirm matches this field. If the fields do not match, it adds an error to display.errors in the Variables panel.

Assimilation

The debugger iterates over each field and processes each <Disable> expression. For those fields whose <Disable> expressions return false, the debugger processes the <Property> objects of the field's <Display> element.

Note that this phase is typically skipped. It is relevant only for certain forms (such as the login form) that do not contain display.mementos. This is necessary for these forms to reconstruct the HTML components during post data assimilation.

HTML Generation (Root Component)

The debugger iterates over only the top level form and the form's <FieldDisplay> element. The goal of this pass is to build the top-level HTML component. HTML Generation (subcomponents) pass follows immediately.

HTML Generation (Subcomponents)

The debugger iterates over each field. It also processes each <Disable> expression. For those fields whose <Disable> expressions return false, the debugger processes the <Property> elements of the field's <Display> element.

Custom View Processing

Certain views require additional passes over the form. During these passes, the debugger iterates over each field and processes each <Disable> expression.

Working with Anonymous Sources

When stepping through forms, the debugger can identify an anonymous source. An *anonymous source* is a form (or portion of a form) that is generated on the fly. As a result, an anonymous source does not correspond to a persistent form that resides in the Identity Manager repository. (Examples of anonymous sources include the login forms and the MissingFields form.)

You cannot set individual breakpoints on an anonymous source because they do not reside in the Identity Manager repository and thus lack a unique identifier.

However, you can step through an anonymous source.

To set a breakpoint on all anonymous sources, select the Global tab in the Breakpoints panel. The debugger subsequently breaks any time it encounters a line from an anonymous source. For example, to debug the login form, select this option, and go to the login page.

Index

SYMBOLS

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X