# Sun™ Identity Manager 8.0 Deployment Tools

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

Part No: 820-2962-10

# 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. This information is organized as follows:



## 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, "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 2, "Developing Custom Adapters" — Describes how to create custom Identity Manager resource adapters that are tailored to your company or customers.
- Chapter 3, "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 4, "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 5, "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.

**NOTE**

- The "Using the Identity Manager IDE" chapter (provided in previous releases) has been removed from this book. Instructions for installing and configuring the Identity Manager Integrated Development Environment (Identity Manager IDE) are now provided on `https://identitymanageride.dev.java.net`.

  For your convenience, instructions for using Identity Manager's Profiler and the Identity Manager FAQ are provided in the "Identity Manager Deployment Tools" section of the "Documentation Additions and Corrections" chapter of the *Sun Java™ System Identity Manager 8.0 Release Notes*.

- The Business Process Editor (BPE) is *deprecated*, and will be removed in the next Identity Manager release. Please use the Identity Manager IDE instead.

# 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.<br>Use `ls -a` to list all files.<br>`% You have mail.` |
| **`AaBbCc123`** (Monospace bold) | What you type, when contrasted with onscreen computer output. | `% su`<br>`Password:` |
| *AaBbCc123* (Italic) | Book titles, new terms, words to be emphasized.<br>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*.<br>These are called *class* options.<br>Do *not* save the file.<br>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 Deployment*: Reference and procedural information that describes how to plan and implement Sun Java™ System Identity Manager Service Provider.
- *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 820-2955-10.

Chapter 1

# Working with Rules

Identity Manager rules and rule libraries are repository objects that are used to encapsulate frequently used programming logic and static variables for reuse in many locations throughout your deployment. Rules and rule libraries enable you to manage data more efficiently.

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



**NOTE** You can use the Identity Manager Integrated Development Environment (Identity Manager IDE) to create, edit, and test rules for your deployment.

Instructions for installing and configuring the Identity Manager IDE are provided on `https://identitymanageride.dev.java.net`.

# Before You Begin

Review the information in these sections before working with Identity Manager rules and rule libraries:



## Intended Audience

This chapter is intended for individuals who create, edit, and test rules for an Identity Manager deployment. Before working with Identity Manager rules and rule libraries, you must

- Have basic programming knowledge
- Understand the XPRESS and XML Object languages

  For detailed information about using XPRESS, see *Sun™ Identity Manager Workflows, Forms, and Views*.
- Understand some Java and Javascript

## Related Documentation and Web Sites

In addition to the information provided in this chapter, consult the publications and web sites listed in this section for information related to working with Identity Manager rules and rule libraries.

### Recommended Reading

See the following chapters in *Sun™ Identity Manager Workflows, Forms, and Views* for information related to Identity Manager rules.

- Chapter 4, “XPRESS Language” describes the XPRESS language.
- Chapter 5, “XML Object Language” describes XML Object syntax.

## Useful Web Sites

The following table describes some web sites you might find useful when trying to work with Identity Manager rules and rule libraries.

**Table 1-1** Useful Web Sites

| Web Site URL | Description |
|---|---|
| https://identitymanageride.dev.java.net | Open source Identity Manager Integrated Development Environment (Identity Manager IDE) project. Includes instructions for installing and configuring Identity Manager IDE. |
| http://sunsolve.sun.com/ | Sun web site containing diagnostic tools, forums, features and articles, security information, and patch contents.<br>**Note**: The information on this site is partitioned into three areas,<br>• Internal: Sun employees only<br>• Contract: Available only to customers with contract access<br>• Public: Available to everyone |
| http://forum.java.sun.com/ | Sun Developer Network (SDN) web site where you can browse forums and post questions. |
| https://sharespace.sun.com/gm/folder-1.11.60181? | Identity Manager link on Sun's Share Space.<br>**Note**: You must sign up for a Share Space ID to access information provided on this site. |
| http://sharespace.sun.com/gm/document-1.26.2296 | Identity Manager FAQ on Sun's Share Space.<br>**Note**: You must sign up for a Share Space ID to access this FAQ. |

# 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 and executing frequently used programming logic or static variables for reuse. Rules are semantically similar to a programming subroutine or function. A rule can take input parameters, execute some logic, and return a value to a caller.

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

Rules are primarily referenced within forms and workflows, but you can also reference rules in other user-data related areas, such as

- **Roles**: Use a role-assignment rule to dynamically assign owners and approvers to a role.
- **Active Sync**: Use Process or Correction rules to control what happens when an Active Sync-enabled adapter detects changes to a resource account.
- **Reconciliation**: Use special rule subtypes (such as confirmation and correlation rules) during reconciliation. These subtypes are described later in this chapter.

---

**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.

For information about writing rules in JavaScript, see "Writing Rules in JavaScript" on page 19.

---

The following example shows how to use the <Rule> element to define a basic rule expression, in which the rule definition name is getApprover, the rule argument name is department, the argument's default value is Tampa, and the rule body returns the Sales Manager or HR Manager string values.

**Code Example 1-1** Example XML Rule

```
<Rule name='getApprover'>
  <Comments> This rule determines the appropriate approver for a particular department.</Comment>
  <RuleArgument name='department'/>
  <RuleArgument name='location' value='Tampa'/>
  <cond>
    <eq><ref>department</ref><String>sales</String></eq>
    <cond>
        <eq><ref>location</ref><String>Tampa</String></eq>
        <String>Tampa Sales Manager</String>
        <String>Sales Manager</String>
    </cond>
    <String>HR Manager</String>
  </cond>
  <MemberObjectGroups>
    ObjectRef type='ObjectGroup' name=ExampleChoc'/>
  </MemberObjectGroups>
</Rule>
```

**NOTE** When *defining* a rule, use the <Rule> element with an uppercase **R** as in <Rule name='rulename'>. When *calling* a rule, use the XPRESS <rule> element with lowercase **r**, as in <rule name='rulename'>.

# Why Use Rules?

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

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

- **Easy maintenance**. You can modify a rule by changing a single object instead of changing each form or workflow that references the rule. You can also more effectively manage
  - Frequently used and shared expressions
  - Frequently changing lists and business logic
- **Distributed development**. Users can develop rules that focus on rule requirements without having to be aware of all forms, Java code, roles, or workflows that reference that rule.
- **Hiding complexity**. More advanced developers can write rules with more complex logic while other users see only the interface without the underlying complexity.

You can secure rules to protect sensitive data, such as user credentials or personal information from being accessed by unauthorized administrators. For more information, see "Securing Rules" on page 28.

## Using Rules in Forms

You typically call a rule in forms to calculate the value of a field 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 rules. Imagine a rule used in a critical form, but the implementation of the rule could be modified by anyIdentity Manager user! For information about securing rules, see *"Securing Rules" on page 28*.

The following example rule returns a list of job titles.

**Code Example 1-2** Returning a Job Titles List

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

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 have to modify this rule instead of modifying each form that references the rule.

In the next example, the `global.jobTitle` field calls the `Job Titles` rule defined in Code Example 1-2 to use the job titles list in a select box:

**NOTE** This example uses a lowercase `r` in the `rule` element because you are calling a rule, not defining a rule.

**Code Example 1-3** 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>
```

Identity Manager forms also support rules that dynamically calculate the name of another rule to call. The following example shows how a form field calls a rule that calculates a department code:

**Code Example 1-4** 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>
```

## Using Rules in Roles

In Identity Manager, a *role* is an object that allows you to efficiently group and assign resources to users. Roles have designated *owners* and *approvers*, where*:*

- Only role owners can authorize changes to the parameters that define the role.
- Only role approvers can authorize the assignment of end-users to the role.

You can directly assign role owners and approvers to a role or use a role-assignment rule to dynamically assign them to a role.

You can use a rule to set the value of any resource attribute in a role definition. When Identity Manager evaluates the rule, it can reference any attribute of the user view.

**NOTE** For more information about roles, see *Sun™ Identity Manager Administration*.

The following example shows how to use a rule to set an attribute value for a particular resource. When you create a user and associate this rule with that user's role, the rule automatically sets the description value.

**Code Example 1-5** Setting the Value for a User's Resource Description

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

## Using Rules in Workflows

In general terms, an Identity Manager *workflow* is a logical, repeatable process during which documents, information, or tasks are passed from one participant to another for action, according to a defined set of procedural rules. A *participant* is a person, machine, or both.

In workflow, you can use a rule anywhere you can use an expression. You can use rules in a workflow to:

- Calculate an approver
- Calculate the name of another rule
- Add a condition to a transition
- Implement an action
- Calculate an approval escalation timeout

For example, you can use a manual action to send an approval request to an administrator, specify a timeout value for this action. If the administrator does not respond within the specified time, you can terminate the action, and escalate the workflow approval to a different administrator.

Workflow activities can also contain subprocesses containing a rule that dynamically calculates a subprocess name. For example.

**Code Example 1-6** Calculating a Rule 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>
```

# 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. Add rules to a rule library when you want to provide a grouping of related functionality. Using libraries simplifies rule maintenance by reducing the number of objects in the Repository. Using libraries also makes it easier to identify and call useful rules when you are designing forms and workflows.

**NOTE** Instructions for invoking rules in a rule library are provided in "Invoking Rules in a Library" on page 21.

The following example shows a library containing two different account ID generation rules:

**Code Example 1-7** 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>
```

**NOTE** You can use the open source Identity Manager Integrated Development Environment (Identity Manager IDE) to view and edit the default rule libraries or to add new rules to an existing library object. See https://identitymanageride.dev.java.net for more information.

# 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 8 and *Sun™ Identity Manager Administration*.
- For information about adding rules to an existing rule library, see "Customizing Default Rules and Rule Libraries" on page 29.
- For information about using XPRESS to write a rule, see the XPRESS Language chapter in *Sun™ Identity Manager Workflows, Forms, and Views*.

➤ **Best Practice:**

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

Identity Manager rules are typically written in XML and encapsulated in the `<Rule>` element.

This section covers the following topics:

- Using the <Rule> Element
- Returning Static Values
- Referencing Variables
- Declaring a Rule with Arguments
- Rules with Side Effects

## Using the <Rule> Element

Code Example 1-8 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 1-8** 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>
```

**NOTE** When *defining* a rule, use the `<Rule>` element with an uppercase **R** as in `<Rule name='rulename'>`. When *calling* a rule, use the XPRESS `<rule>` element with lowercase **r**, as in `<rule name='rulename'>`.

## Returning Static Values

If the rule returns a static value, you can write it using XML Object syntax. The following example returns a list of strings.

**Code Example 1-9** 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™ Identity Manager Workflows, Forms, and Views*.

## Referencing Variables

You can use `<ref>` expressions in a rule to reference the values of external variables. The context in which the rule is used determines the names of the available variables.

- In forms, you can reference any form field, view attribute, or variable defined with `<defvar>`.
- In workflows, you can reference any variable defined within the workflow process.

In the following example, the 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 1-10** 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>
```

The next example shows how a workflow uses a rule to test whether a transition to a particular activity should be taken. This workflow defines a `user` variable that contains the User view. The rule returns `true` if any simulated resources are assigned to this user or returns `null` if no simulated resources are assigned. The workflow engine interprets `null` as false and would consequently not take the transition.

**Code Example 1-11** Testing a Transition

```
<Rule name='Has Simulated Resources'>
   <notnull>
      <ref>user.accountInfo.types[simulated].accounts</ref>
   </notnull>
</Rule>
```

## Declaring a Rule with Arguments

> ➤ **Best Practice:**
>
> You are not required to declare arguments for a rule, but it is considered a best practice to do so. If a rule uses a variable that is "in scope" at the time of the rule's execution, then the rule becomes less reusable.

Declaring arguments in a rule provides documentation to rule users, allows reference validation in the Identity Manager IDE, and allows the rule to be used in forms and workflows that might not use the same naming convention.

You can use the `<RuleArgument>` element to declare rule arguments, and set a default value for the argument by specifying a `value` after the argument name. For example, the following rule specifies "`Austin`" as the default value for the `location` RuleArgument.

**Code Example 1-12** 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>
```

You can use this rule in user forms, but `UserId` and `location` are not attributes of the User view. You must use the `<argument>` element in the rule call to pass the expected arguments into the rule. Note that passing an argument whose name is `location` overrides the default value declared in the `RuleArgument` element in the rule definition.

**Code Example 1-13** 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 19.

There is no formal way to declare an argument type, but you can specify type in a comment field. Use the `<Comment>` element to include comments in your rule:

**Code Example 1-14** 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 or to take an action other than returning a value. You can use the following XPRESS expressions in a rule to assign values to external variables:

- `<setvar>`: Use to specify a variable value.
- `<setlist>` : Use to assign a value into a specified position in a list, overwriting the current value.
- `<putmap>` Use to specify map elements to an object.

The following example shows how the rule tests the value of external variable named `department` and assigns values to two other variables.

**Code Example 1-15** 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 might find it more convenient to write those rules in JavaScript rather than XPRESS, and then wrap the JavaScript in an XPRESS `<script>` element.

The following example references the values of form and workflow variables, calls the `env.get` function, and passes the variable name. The example uses the `env.put` function to assign variable names, and the value of the last statement in the script becomes the value of the rule. The rule returns the value in the `email` variable.

**Code Example 1-16** 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>
```

You can call other rules with the `env.call` function.

# Referencing Rules

This section provides information about referencing rules. The information is organized as follows:

- Basic Rule Call Syntax
- Invoking Rules in a Library
- Resolving Rule Arguments

# 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, the interpreter assumes the value of the name attribute is the name of a rule object in the repository. The interpreter automatically loads the rule from the repository and evaluates that rule. 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 uses an argument element to pass an accountId argument to the rule. In addition, the argument value is passed as a static string, jsmith.

```
<rule name='getEmployeeId'>
   <argument name='accountId' value='jsmith'/>
</rule>
```

You can also use an expression to calculate the value of an argument, as follows. In this example, the argument value is calculated by evaluating a simple <ref> expression that returns the value of the view attribute user.waveset.accountId.

```
<rule name='getEmployeeId'>
   <argument name='accountId'>
      <ref>user.waveset.accountId</ref>
   </argument>
</rule>
```

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>
```

Both of the previous examples pass the value of the user.waveset.account view attribute as the value of the argument.

# Invoking Rules in a Library

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, each rule name 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'/>
```

# Resolving Rule Arguments

Most rules contain XPRESS `<ref>` expressions or JavaScript `env.get` calls to retrieve variable values. Several options are available for controlling how the values of these variables are obtained.

In the simplest case, the application calling the rule attempts to resolve all references.

- For rules called from forms, the form processor assumes all references are to attributes in a view.
- For rules called from workflows, the workflow processor assumes all references are to workflow variables.
- Rules can call other rules by dynamically resolving the called rule's name. You can use the optional `<RuleArgument>` element, which is described in "Declaring a Rule with Arguments" on page 16.

This section provides the following information:

## Calling Scope or Explicit Arguments in Forms

This section provides examples that illustrate how rule arguments are resolved in forms.

The following example shows how to add a rule to a form. You can use this form with the User view because there are attribute names in the view.

```
<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 the following example:

**Code Example 1-17** 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.

The following example shows a modified version of the rule that references the variables `firstname` and `lastname`:

**Code Example 1-18** Rule Referencing firstname and lastname Variables

```
<Rule name='generateEmail'>
<RuleArgument name='firstname'/>
<RuleArgument name='lastname'/>
   <concat>
      <ref>firstname</ref>
      <s>.</s>
      <ref>lastname</ref>
      <s>@example.com</s>
   </concat>
</Rule>
```

The following examples shows a rule that is simpler and more general. The example does not assume that the rule will be called from a user form, but that the rule must be called with explicit arguments.

**Code Example 1-19** 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, which keeps the rule isolated from the naming conventions used by the calling application and makes the rule usable in other contexts.

## Using the LocalScope Option in Workflows

Even when arguments are passed explicitly to a rule, the system by default allows references to other variables that are not passed as explicit arguments. The following example shows a workflow action calling the rule but passing only one argument:

**Code Example 1-20** 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 is 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 1-21** 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 must use the `localScope` option.

## Using Rule Argument Declarations

> ➤ **Best Practice:**
>
> You are not required to include explicit declarations for all arguments that can be referenced by a rule within the rule definition, but it is considered a best practice to do so.

Using argument declarations offers the following advantages:

- Declarations can serve as documentation for the caller of the rule
- Declarations can define default values
- Declarations can enable the Identity Manager IDE to check for misspelled references within the rule
- Declarations can enable the Identity Manager IDE to simplify the configuration of a rule call

For example, you could rewrite the `generateEmail` rule as follows:

**Code Example 1-22** 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.

In this example, the rule was modified to define another argument named `domain`, which was given a default value of `example.com`. This rule uses the default value unless the caller passes an explicit argument named `domain`.

The next example shows a call that produces the `john.smith@example.com` string:

**Code Example 1-23** Producing john.smith@example.com String

```
<rule name='generateEmail'>
   <argument name='firstname' value='john'/>
   <argument name='lastname' value='smith'/>
</rule>
```

The next example shows a call that produces the `john.smith@yourcompany.com` string:

**Code Example 1-24** 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>
```

This example shows a call that produces the `john.smith@` string:

**Code Example 1-25** 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`.

## Using Locked Arguments

Declaring arguments with default values can be a useful technique for simplifying the development and customization of rules. If you have a constant value in a rule that might occasionally change, it is easier to locate and change that value if it is defined in an argument rather than embedded deep within a rule expression.

The Identity Manager IDE provides a simplified user interface for configuring rules. You can change the default values of arguments in the Identity Manager IDE, which is much easier than editing the entire rule expression.

After an argument is declared, it is possible for the caller of the rule to override the default value by passing an explicit argument. However, if you do not want the caller to have any control over the argument value, include a `locked` attribute with a value of `true` in the `RuleArgument` element to lock the argument. For example,

**Code Example 1-26** 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>
```

The `domain` argument is locked in this example, which means the argument value will always be `example.com` — even if the caller tries passing a value for the argument. If you are going to use this rule at a site where the domain name is not `example.com`, the administrator only has to edit the rule to change the argument value. The administrator does not have to understand or modify the rule expression.

# Securing Rules

If a rule contains sensitive information, such as credentials or calls to a Java utility that might have dangerous side effects, you must secure the rule to prevent anyone from using that rule in an unintended way.

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 who is capable of creating a session through the API or a SOAP request.

This section provides the following information:

- Put Rules in an Appropriate Organization
- Use Authorization Types to Secure Rules
- Control Access to Rules that Reference More Secure Rules

## Put Rules in an Appropriate Organization

As a convenience, most administrators put simple rules, such as those that perform calculations but have no side effects, in the `All` organization so that everyone granted rights to view rules can access those rules.

However, if you want to provide more security for a rule

- Do not put sensitive rules in the `All` organization.
- Put the rule in an appropriate organization such as `Top` (or another suitably high-level organization) so that only high-level administrators can execute that rule directly.

## Use Authorization Types to Secure Rules

You can use authorization types (`AuthType`) to further scope or restrict access to a subset of objects for a given Identity Manager objectType, such as a rule. For example, you might not want your users to have access to all rules within their scope of control when populating rules to select in a user form.

For information about using authorization types, see “Using Authorization Types to Secure Objects“ in *Sun™ Identity Manager Administration*.

## Control Access to Rules that Reference More Secure Rules

Users can call, view, and modify the content of a secure rule if they have been given access to a rule that references that secure rule.

Identity Manager runs an authorization check in which a wrapper calls all of the users who have a right to edit that rule. Authorized users can use that rule to call other rules without further authorization checking, which can give them indirect access to secure rules.

When you create a rule that references a secure rule and give users access rights to the less secure rule, be careful that you are not inadvertently giving them inappropriate access to the secure rule.

**NOTE** To create a rule that references a more secure rule, you must control both organizations containing those rules. You also must have rights to run the first rule and call the secure rule.

# Customizing Default Rules and Rule Libraries

This section describes the default rules and rule libraries supplied with Identity Manager. The information is organized as follows:

- Identity Manager Rules
- Auditor Rules
- Audit Policy Rules
- Service Provider Rules

**NOTE** You can use the Identity Manager IDE to customize these rules and rule libraries.

# Identity Manager Rules

You can use the following rules and rule libraries to customize Identity Manager.

- AccessEnforcerLibrary
- ActiveSync Rules
- ADRules Library
- AlphaNumeric Rules Library
- Approval Transaction Message
- Approval Transaction Message Helper
- Attestation Remediation Transaction Message
- Attestation Remediation Transaction Message Helper
- Attestation Transaction Message
- Attestation Transaction Message Helper
- CheckDictionaryWord
- DateLibrary
- End User Controlled Organizations
- EndUserRuleLibrary
- ExcludedAccountsRule
- getAvailableServerOptions
- InsertDictionaryWord
- Is Manager
- LoginCorrelationRules
- My Direct Reports
- NamingRules Library
- NewUsernameRules
- Object Approvers As Attestors
- Object Owners As Attestors
- Organization Names

- OS400UserFormRules
- IS_DELETE
- RACFUserFormRules
- Reconciliation Rules
- RegionalConstants Library
- Remediation Transaction Message
- Remediation Transaction Message Helper
- ResourceFormRules
- Resource Names
- Role Approvers
- Role Notifications
- Role Owners
- Sample On Local Network
- SAP Portal User Form Default Values
- ShellRules
- SIEBEL_NAV_RULE
- TestDictionary
- TopSecretUserFormRules
- User Members Rule
- USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CONF
- USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR
- USER_FIRST_AND_LAST_NAMES_MATCH_ACCOUNT
- USER_NAME_MATCHES_ACCOUNT_ID
- USER_OWNS_MATCHING_ACCOUNT_ID
- Users Without a Manager
- Use SubjectDN Common Name

## AccessEnforcerLibrary

The AccessEnforcerLibrary is a default library of rules that enable you to manage certain types of objects because the Access Enforcer resource adapter does not provide a way for you to fetch these objects.

**Inputs:** See Table 1-2.

You must specify the following for a custom AccessEnforcerLibrary rule:

**AuthType:** `Library`

**SubType:** `listRules`

**Returns:** See Table 1-2

**Predefined Rules:** Not specified

The following table describes the example AccessEnforcerLibrary rules.

**Table 1-2** Example AccessEnforcerLibrary Rules

| Rule Name | Input Variables | Description |
|---|---|---|
| getApplications | • `resName` (Resource name)<br>• Specify Access Enforcer object names by manually entering the names as strings. | Returns a list of applications that are available in SAP GRC Access Enforcer. If `resName` was specified, fetches the applications from Access Enforcer. Otherwise, returns the list specified statically. |
| getRoles | `resName` (Resource name) | Returns a list of roles that are available in SAP GRC Access Enforcer that are the same as the roles available in the back-end system.<br>These values are manually created and must be in sync with the corresponding values in SAP GRC Access Enforcer. |
| getRequestTypes | None | Returns a list of Request types that are available in SAP GRC Access Enforcer.<br>These values are manually created and must be in sync with the corresponding values in SAP GRC Access Enforcer. |
| getPriorities | None | Returns a list of Priority values that are available in SAP GRC Access Enforcer<br>These values are manually created and must be in sync with the corresponding values in SAP GRC Access Enforcer. |

**Table 1-2** Example AccessEnforcerLibrary Rules

| Rule Name | Input Variables | Description |
|---|---|---|
| getEmployeeTypes | None | Returns a list of Employee types that are available in SAP GRC Access Enforcer.<br><br>These values are manually created and must be in sync with the corresponding values in SAP GRC Access Enforcer. |
| getSLAs | None | Returns a list of Service Levels that are available in SAP GRC Access Enforcer.<br><br>These values are manually created and must be in sync with the corresponding values in SAP GRC Access Enforcer. |
| getSupporttedVersions | `resName` (Resource name) | Returns a list of SAP GRC Access Enforcer versions that are supported by Identity Manager. These values must be the same as values that the adapter facet understands. |

## ActiveSync Rules

When the Flat File 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. The adapter uses process, correlation, and delete rules to determine what to do with the user.

**NOTE** Active Sync rules must use `context`, not `display.session`. Correlation and Delete rules do not get a session, but Confirmation rules do. For more information, see "Correlation Rule" on page 61 and "Confirmation Rule" on page 62.

**Inputs:** Accepts resource account attributes in the `activeSync` namespace. For example, `activeSync.firstname`.

You must specify the following for a custom ActiveSync rule:

**AuthType:** Not specified

**SubType:** Not specified

**Namespace:** Provide resource account attributes in the `activeSync` namespace. For example,

```
activeSync.firstname
```

**Predefined Rules:** ActiveSyncRules' predefined rules include:

- **ActiveSync has isDeleted set**: Used by migration from resources when you set the `Process deletes as updates` parameter to `false`.

  **NOTE** Do not change this rule name. If you want to use a different rule name, duplicate the rule content and rename the new rule.

- **No Correlation Rule**: Use this default rule if you do not want correlation.
- **No Confirmation Rule**: Use this default rule if you do not want confirmation.

## ADRules Library

The default library of ADRules enables you to create a list of the servers

**Inputs:** None

You must specify the following for a custom ADRules rule:

**AuthType:** Not specified

**SubType:** Not specified

**Called:**

**Returns:** A list of zero or more string values.

**Predefined Rule:** None

**Table 1-3** Example ADRules Rules

| Rule Name | Description |
|---|---|
| Exchange Servers | Returns a list of the Exchange servers in your environment.<br>You can update this list to include the Exchange servers in your environment. |
| Home Directory Servers | Returns a list of the Home Directory Servers in your environment.<br>You can update this list to include the systems that serve home directory drives in your environment. |
| AD Login Scripts | Returns a list of the user login scripts being used in your environment.<br>You can update this list to include the login batch scripts in your environment. |
| Home Directory Drive Letter | Returns a list of the home directory mapped drive letters in your environment.<br>You can update this list to include the common home directory map drive letters in your environment. |
| Home Directory Volumes | Returns a list of the home directory volume names in your environment.<br>You can update this list to include the common home directory volume names in your environment. Identity Manager uses this value with the Home Directory Server to create a user's home directory. This volume must exist and be shared on the selected home directory server. |

## AlphaNumeric Rules Library

The AlphaNumeric Rules Library is a default library of rules that enable you to control how numbers and letters are ordered and displayed in Identity Manager forms and workflows.

**NOTE** This library is displayed as the `Alpha Numeric Rules` library object in the Identity Manager IDE.

**Inputs:** See Table 1-4.

You must specify the following for a custom rule:

**AuthType:** `EndUserRule`

**SubType:** Not specified

**Returns:** A list of zero or more strings

The following table describes rules in the AlphaNumeric Rules library.

**Table 1-4** Example Alphanumeric Rules

| Rule Name | Input Variable | Description |
|---|---|---|
| AlphaCapital | None | Returns a list of English capital alpha characters |
| AlphaLower | None | Returns a list of English lowercase alpha characters |
| Numeric | None | Returns a list of numeric characters |
| WhiteSpace | None | Returns a list of white space characters |
| SpecialCharacters | None | Returns a list of common special characters |
| legalEmailCharacters | None | Returns a list of legal special characters for email |
| stringToChars | `testStr` | Converts the given string to a list composed of the string's individual characters |
| isNumeric | `testStr` | Tests to see if `testStr` contains all numeric characters |
| isAlpha | `testStr` | Tests to see if `testStr` contains only alpha characters |
| hasSpecialChar | `testStr` | Tests to see if `testStr` contains any special characters |
| hasWhiteSpace | `testStr` | Tests to see if `testStr` contains any white space characters |
| isLegalEmail | `testStr` | Tests to see if `testStr` consists of only legal email address characters |
| StripNonAlphaNumeric | `testStr` | Removes any non-alpha or non-numeric characters from `testStr` |

## Approval Transaction Message

The Approval Transaction Message rule is a default rule used to format approval transaction text. You can customize this rule to provide more information for a user to sign.

**Inputs:** Accepts the following arguments:

- `workItemList`: A set of `workitems` that are being approved.
- `variablesList`: A set of variables corresponding to each `workitem` in `workitemList`.
- `approverName`: User being asked to approve the `workitems`.

You must specify the following for a custom Approval Transaction Message rule:

**AuthType:** Not Specified

**SubType:** Not Specified

**Returns:** Formatted transaction text for the list of `workitems` in `workItemList`

**Predefined Rule:** None

## Approval Transaction Message Helper

The Approval Transaction Message Helper rule returns the formatted transaction text for the approval of a single `workitem`.

**Inputs:** Accepts the following arguments:

- `workItem`: The `workitem` that is being approved.
- `variables`: The `workitem` variables.

You must specify the following for a custom Approval Transaction Message Helper rule:

**AuthType:** Not Specified

**SubType:** Not Specified

**Returns:** Formatted transaction text for the approval of a single `workitem`

**Predefined Rule:** None

## Attestation Remediation Transaction Message

The Attestation Remediation Transaction Message rule is a default rule used to format attestation remediation transaction text. You can customize this rule to provide more information for the user to sign.

**Inputs:** Accepts the following arguments:

- `workItemList`: A set of `workitems` that are being approved.
- `variablesList`: A set of variables corresponding to each `workitem` in `workitemList`.
- `approverName`: User being asked to approve the `workitems`.
- `action`: Expected to be `remediate`.
- `actionComments`: Comments that are entered as part of the remediation.

You must specify the following for a custom Attestation Remediation Transaction Message rule:

**AuthType:** `EndUserAuditorRule`

**SubType:** Not Specified

**Returns:** Formatted attestation remediation transaction text

**Predefined Rule:** None

## Attestation Remediation Transaction Message Helper

The Attestation Remediation Transaction Message Helper rule returns the formatted transaction text for the attestation remediation of a single `workitem`.

**Inputs:** Accepts the following arguments:

- `workItem`: The `workitem` that is being approved.
- `variables`: The `workitem` variables.

You must specify the following for a custom Attestation Remediation Transaction Message Helper rule:

**AuthType:** `EndUserAuditorRule`

**SubType:** Not Specified

**Called:**

**Returns:** Formatted transaction text for the attestation remediation of a single `workitem`.

**Predefined Rule:** None

## Attestation Transaction Message

The Attestation Transaction Message rule a default rule used to format attestation transaction text. You can customize this rule to provide more information for the user to sign.

**Inputs:** Accepts the following arguments:

- `workItemList`: A set of `workitems` that are being approved.
- `variablesList`: A set of variables corresponding to each `workitem` in `workitemList`.
- `approverName`: User being asked to approve the `workitems`.
- `action`: Expected to be `aproved` or `approve`.
- `actionComments`: Comments that are entered as part of the attestation.

You must specify the following for a custom Attestation Transaction Message rule:

**AuthType:** `EndUserAuditorRule`

**SubType:** Not Specified

**Called:**

**Returns:** Formatted attestation transaction text

**Predefined Rule:** None

## Attestation Transaction Message Helper

The Attestation Transaction Message Helper rule returns the formatted transaction text for the a single attestation.

**Inputs:** Accepts the following arguments:

- `workItem`: The `workitem` that is being approved.
- `variables`: The `workitem` variables.

You must specify the following for a custom Attestation Transaction Message Helper rule:

**AuthType:** `EndUserAuditorRule`

**SubType:** Not Specified

**Called:**

**Returns:** Formatted transaction text for the a single attestation

**Predefined Rule:** None

## CheckDictionaryWord

Use the CheckDictionaryWord rule to run a JDBC query against a dictionary to check if a password exists in the dictionary.

**Inputs:** Accepts the following arguments:

- `type`
- `driverClass`
- `driverPrefix`
- `url`
- `host`
- `port`
- `database`
- `context`
- `user`
- `password`

- `sql`
- `arg1`

You must specify the following for a custom CheckDictionaryWord rule:

**AuthType:** Not Specified

**SubType:** Not Specified

**Called:**

**Returns:** A list of zero or more strings.

**Predefined Rule:** None

## DateLibrary

The DateLibrary is a default library of rules that control how dates and times are displayed in a deployment.

**NOTE** This library is displayed as the `Date Library` library object in the Identity Manager IDE.

**Inputs:** See Table 1-5.

You must specify the following for a custom DateLibrary rule:

**AuthType:** `Rule`

**SubType:** Not specified

**Returns:** Boolean values of `true` or `false`. See Table 1-5.

The following table describes the example DateLibrary rules.

**Table 1-5** Example DateLibrary Rules

| Rule | Input Variables | Description |
|---|---|---|
| `Date Validation` | `mm/dd/yy yy` | Determines valid date strings.<br>If month or day values are provided in with single digits, the rule accounts for them appropriately.<br>• `true` if the string provided contains valid date components.<br>• `false` if the string provided contains invalid date components. |
| `Validate Day Month Year` | • `month`<br>• `day`<br>• `year` | Determines valid day, month, and year strings. If the month or the day values are provided in with single digits, the rule accounts for them appropriately.<br>• `true` if the string provided is a valid date.<br>• `false` if the string provided is a invalid date. |
| `Validate Time` | `HH:mm:ss` | Determines valid time strings.<br>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 returns a `false`.<br>• `true` if the string provided is a valid time.<br>• `false` if the string provided is a invalid time. |

## End User Controlled Organizations

The End User Controlled Organizations rule determines the set of organizations that are controlled by a user logging into the End User interface. These organizations, together with the End User organization, define the scope of control over which a user is granted the permissions specified in the `EndUser` capability (`AdminGroup`). Because this is a rule, it allows the scope of control to vary depending on which user is logging into the End User interface.

**Inputs:** User view of the authenticating end user

You must specify the following for a custom End User Controlled Organizations rule:

**AuthType:** `EndUserControlledOrganizationsRule`

**SubType:** Not Specified

**Returns:** A single controlled organization (`string`) or a list of controlled organizations. Each value can be an organization name or ID. If an organization name is returned, it must be fully qualified up to Top (for example, `Top:Marketing:South`)

**Predefined Rule:** Defaults to returning the organization of which the user is a member (for example, `waveset.organization`)

### EndUserRuleLibrary

The EndUserRuleLibrary is a default library of rules that Identity Manager uses to determine or to verify end-user account information.

**NOTE** By default, Identity Manager's End User Anonymous Enrollment processing generates values for `accountId` and `emailAddress` by using user-supplied first names (`firstName`), last names (`lastName`) and employee IDs (`employeeID`). Anonymous enrollment can cause non-ASCII characters to display in email addresses and account IDs.

To ensure that Identity Manager maintains ASCII accountIds and email addresses during anonymous enrollment processing, international users must perform these steps:

1. Modify the following EndUserRuleLibrary rules:
   - `getAccountId`: Remove `firstName`, `lastName`, and `letter substr`. Use `employeeId` only.
   - `getEmailAddress`: Remove `firstName`, `lastName`, and `"."` Use `employeeId` only.
   - `verifyFirstname`: Change length check from `2` to `1` to allow single character Asian first names.
2. Edit the End User Anon Enrollment Completion form to remove the `firstName` and `lastName` arguments from calls to the `getAccountId` and `getEmailAddress` rules.

**NOTE** This library is displayed as the `EndUserRuleLibrary` library object in the Identity Manager IDE.

**Inputs:** See Table 1-6 and Table 1-7.

You must specify the following for a custom EndUserLibrary rule:

**AuthType:** `EndUserLibrary`

**SubType:** Not specified

The following table describes the example EndUserRuleLibrary rules.

**Table 1-6** Example EndUserRuleLibrary Rules

| Rule | Input Variable | Description |
|---|---|---|
| `getCallerSession` | None | Returns the internal session context (Lighthouse context) for the user executing a form. |
| `getUserView` | • `resourceTargets` list<br>• `accountId` string<br>• `includeAvailableRoleInfos` boolean | Returns the User view of the specified accountId, including a list of resource targets, and whether or not to include Role information. |
| `getView` | • `nameOrId` string<br>• `type` string<br>• `options` map | Returns a view of an object specified by the name or GUID, type of object, and a map of options. |
| `getUnassignedResources` | • `roles` list<br>• `currentResources` list<br>• `groups` list | Determines which resources are currently unassigned. |
| `getDirectReports` | • `manager` string<br>• `options` map | Returns a list of direct reports for a specified manager. For example, a list of users whose `idmManager` attribute is specified by the `manager` input variable. |
| `getIndirectReports` | • `manager` string<br>• `options` map | Returns a list of indirect reports for a specified manager. For example, a list of users who are in the reporting structure of the user specified by the `manager` input variable, excluding direct reports. |
| `getResourceObject ParentId` | • `resourceName` string<br>• `resObjectName` string<br>• `objType` string<br>• `objAttr` string | Returns a GenericObject of the parent of a resource specified by the name, object type, and object attribute. |
| `getObjectsByType` | • `type` string<br>• `attributeVal` string<br>• `attributeName` string | Returns a list of GenericObjects specified by type and that match the `attributeName=attributeVal` condition. |

**Table 1-6** Example EndUserRuleLibrary Rules *(Continued)*

| Rule | Input Variable | Description |
|---|---|---|
| `getRealName` | • `accountId` string<br>• `addAccountId` boolean | Determines a user's "real name," such as FirstName <space> LastName, when an `accountId` has been provided.<br>• If the `addAccountId` argument is `true`, Identity Manager returns the `FirstName LastName (accountId)` string.<br>• If the `FirstName` or `LastName` attributes cannot be determined, the rule returns just the `accountId`.<br>**NOTES**:<br>• You can easily modify this rule if you want the real name to display as `LastName`, `FirstName`.<br>• The user must have the appropriate permissions to be able to search for other users. |

The next table describes the example EndUserRuleLibrary rules used for anonymous enrollment.

**Table 1-7** Example EndUserRuleLibrary Rules for Anonymous Enrollment

| | | |
|---|---|---|
| `getAccountId` | • `firstName` string<br>• `lastName` string<br>• `employeeId` string | Generates an account ID from the first name, last name, and employee ID.<br>`First initial + last intial + employee ID`<br>**Note**: International users must modify this rule to ensure that Identity Manager maintains ASCII accountIds and email addresses during anonymous enrollment processing. See page 44 for instructions. |
| `getEmailAddress` | • `firstName` string<br>• `lastName` string<br>• `emailDomain` string | Generates an email address from the first name, last name, and email domain provided.<br>`firstname.lastname@emailDomain`<br>**Note**: International users must modify this rule to ensure that Identity Manager maintains ASCII accountIds and email addresses during anonymous enrollment processing. See page 44 for instructions. |

**Table 1-7** Example EndUserRuleLibrary Rules for Anonymous Enrollment*(Continued)*

| | | |
|---|---|---|
| `getIdmManager` | `employeeId` string | Returns the account ID of the Identity Manager manager associated with an employee ID for a user being created. You must customize this rule for your deployment environment. (Default is *configurator*.) |
| `getOrganization` | None | Returns the name of the organization to which a user will be assigned. You must customize this rule for your deployment environment. (Default is *Top*.) |
| `runValidation` | None | Invokes `verifyFirstname`, `verifyLastname`, `verifyEmployeeId`, and `verifyEligibility` rules. |
| `verifyFirstname` | `firstName` string | Validates the first name provided by a user for the End User Anonymous Enrollment process. This sample rule verifies a first name is not null. You must customize this rule for your deployment environment.<br>**Note**: International users must modify this rule to ensure that Identity Manager maintains ASCII accountIds and email addresses during anonymous enrollment processing. See page 44 for instructions. |
| `verifyLastname` | `lastName` string | Validates the last name provided by a user for the End User Anonymous Enrollment process. This sample rule verifies a last name is not null. You must customize this rule for your deployment environment. |
| `verifyEmployeeId` | `employeeId` string | Validates the employee ID provided by a user for the End User Anonymous Enrollment process. This sample rule verifies that an employee ID is valid. You must customize this rule for your deployment environment. |
| `verifyEligibility` | • `firstName` string<br>• `lastName` string<br>• `employeeId` string | Can be used to validate the employee ID provided by a user for the End User Anonymous Enrollment process. This rule must be customized for deployment. |

## ExcludedAccountsRule

The `ExcludedAccountsRule` supports the exclusion of resource accounts from resource operations.

**Inputs:** Accepts the following arguments:

- `accountId`: String account ID being tested.

  You can compare the `accountId` argument 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` argument 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, every account identified by the rule is excluded from all of 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)

You must specify the following for a custom ExcludedAccountsRule rule:

**AuthType:** `ExcludedAccountsRule`

**SubType:** Not specified

**Predefined Rules:**

- Microsoft SQL Server Excluded Resource Accounts
- Sun Access Manager Excluded Resource Accounts
- Unix Excluded Resource Accounts
- Windows Excluded Resource Accounts

The following example exemplifies subType use and excludes specified resource accounts for UNIX adapters.

**Code Example 1-27** Exemplifying `authType` Use

```
<Rule name='ExcludedResourceAccounts' 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>
```

The next example shows how to use the `operation` parameter. This parameter allows you to manipulate the "Test User" resource account — without impacting Identity Manager — if Active Sync is running against the resource.

**Code Example 1-28** Example Using `operation` Parameter

```
<Rule name='Example Excluded Resource Accounts' authType='ExcludedAccountsRule'>
<!--
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>
```

**Code Example 1-28** Example Using `operation` Parameter *(Continued)*

```
         <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>
            </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>
```

This example shows an ExcludedAccountsRule for RACF.

**Code Example 1-29** ExcludedAccountsRule for RACF

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Waveset PUBLIC "waveset.dtd" "waveset.dtd">
<Waveset>
 <Rule name="RACF EAR" authType="ExcludedAccountsRule">
   <RuleArgument name="accountID"/>
   <block>
     <defvar name="excludedList">
       <List>
         <String>irrcerta</String>
         <String>irrmulti</String>
         <String>irrsitec</String>
         <String>IBMUSER</String>
       </List>
     </defvar>
     <cond>
       <eq>
         <containsAny>
           <ref>excludedList</ref>
           <list>
             <upcase>
               <ref>accountID</ref>
             </upcase>
             <ref>accountID</ref>
           </list>
         </containsAny>
         <i>1</i>
       </eq>
       <Boolean>true</Boolean>
       <Boolean>false</Boolean>
     </cond>
   </block>
   <MemberObjectGroups>
     <ObjectRef type="ObjectGroup" id="#ID#Top" name="Top"/>
   </MemberObjectGroups>
 </Rule>
</Waveset>
```

This final example shows an ExcludedAccountsRule for RACF LDAP.

**Code Example 1-30** Excluded Accounts Rule for RACF LDAP

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Waveset PUBLIC "waveset.dtd" "waveset.dtd">
<Waveset>
<Rule name="Test RACF_LDAP Case Insensitive Excluded Resource Accounts"
authType="ExcludedAccountsRule">
 <RuleArgument name="accountID"/>
 <block>
   <defvar name="excludedList">
     <List>
       <String>irrcerta</String>
       <String>irrmulti</String>
       <String>irrsitec</String>
       <String>IBMUSER</String>
     </List>
   </defvar>
   <defvar name="convertedId">
     <get>
       <split>
         <get>
           <split>
             <ref>accountID</ref>
             <s>,</s>
             </split>
           <i>0</i>
         </get>
         <s>=</s>
       </split>
       <i>1</i>
     </get>
   </defvar>
   <cond>
     <eq>
       <containsAny>
         <ref>excludedList</ref>
         <list>
           <upcase>
             <ref>convertedId</ref>
           </upcase>
           <ref>convertedId</ref>
```

## getAvailableServerOptions

The getAvailableServerOptions rule determines the list of available server configuration options for the specified synchronization mechanism. Using the settings in `Waveset.properties` applies only for ActiveSync, and is a backwards-compatibility option.

**Inputs:** Accepts the `targetObjectType` argument

**NOTE** If `IDMXUser`, then `viaWavesetProperties` is not returned in the list.

You must specify the following for a custom getAvailableServerOptions rule:

**AuthType:** Not Specified

**SubType:** Not Specified

**Predefined Rule:** None

## InsertDictionaryWord

Use the InsertDictionaryWord rule to run a JDBC command against the Identity Manager dictionary to load new words into the database.

**Inputs:** Accepts the following arguments:

- `type`
- `driverClass`
- `driverPrefix`
- `url`
- `host`
- `port`
- `database`
- `context`
- `user`
- `password`
- `sql`
- `arg1`
- `argList`

You must specify the following for a custom InsertDictionaryWord rule:

**AuthType:** Not Specified

**SubType:** Not Specified

**Called:**

**Returns:** A list of zero or more strings.

**Predefined Rule:** None

## IS_DELETE

The IS_DELETE rule is a sample rule, written for the PeopleSoft Active Sync adapter, that determines whether the Active Sync event should delete a user.

**Inputs:** None

You must specify the following for a custom IS_DELETE rule:

**AuthType:** Not specified

**SubType:** Not specified

**Predefined Rule:** None

## Is Manager

The Is Manager rule tests specified `accountIds` to see whether they are managers for any other users in the system.

**Inputs:** Accepts the `managerId` argument (`<RuleArgument name='managerId'/>`)

You must specify the following for a custom Is Manager rule:

**AuthType:** `RoleConditionRule`

**SubType:** Not Specified

**Called:**

**Returns:** True if `managerId` is declared as the `idmManager` for any user in the system, otherwise returns `false`.

This rule issues a query in the repository using the caller's `display.session` session, meaning this rule can only be called from a Form. The check only matches users that are within organizations controlled by the caller, so the rule might return `false` if the `managerId` is the manager of users outside of the callers scope of control.

**Predefined Rule:** None

## LoginCorrelationRules

The LoginCorrelationRules map user login information to an Identity Manager user. You specify logic in LoginCorrelationRules that enables the rule to search for an Identity Manager user and return a list of one or more `AttributeConditions`.

**Inputs:** None

You must specify the following for a custom LoginCorrelationRules rule:

**AuthType:** `LoginCorrelationRule`

**SubType:** Not specified

**Called:** By a LoginModule to map login information to the Identity Manager user

**Returns:** A list of one or more `AttributeConditions`

**Predefined Rules:**

- Correlate via X509 Certificate SubjectDN
- Correlate via LDAP Uid

## My Direct Reports

The My Direct Reports rule returns the names of all Identity Manager users that are direct reports of the caller. Management is typically a hierarchical structure, however this rule only returns the names of users that have the caller specified as their manager. The management hierarchy is not traversed by this rule.

**Inputs:** None

You must specify the following for a custom My Direct Reports rule:

**AuthType:** `AccessScanRule`

**SubType:** `USER_SCOPE_RULE`

**Called:**

**Returns:** A list of Identity Manager user names that have the caller specified as their manager.

**Predefined Rule:** None

## NamingRules Library

The NamingRules Library is a default library of rules that enable you to control how names are displayed after rule processing.

---

**NOTE** This library is displayed as the `NamingRules` library object in the Identity Manager IDE.

---

**Inputs:** None

You must specify the following for a custom NamingRulesLibrary rule:

**AuthType:** Not specified

**SubType:** Not specified

**Called:**

**Returns:**

**Predefined Rule:** None

The following table lists the example NamingRules.

**Table 1-8** Example NamingRules

| 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<br>**Note**: You must append an `AccountName` rule to the mail domain. |
| Fullname — First space Last | Marcus Aurelius |
| Fullname — First space MI space Last | Marcus A Aurelius |
| Fullname — Last comma First | Aurelius, Marcus |

## NewUsernameRules

The NewUsernameRule is a standard repository initialization file that you can use to extract the value of a user distinguished name's (DN) left most relative distinguished name (RDN).

**Inputs:** None

You must specify the following for a custom NewUsernameRules rule:

**AuthType:** `NewUserNameRule`

**SubType:** Not specified

**Called:**

**Returns:** A proposed user name for new users upon registration. For example, `Use SubjectDN Common Name` extracts the `jsmith` from `cn=jsmith,ou=engineering,dc=acme,dc=com`.

**Predefined Rule:** Use SubjectDN Common Name

## Object Approvers As Attestors

The Object Approvers As Attestors rule returns the provided `objectapprovers` parameter value if it is not null. If the `objectapprovers` list is not provided, this rule creates a new list and includes the `Configurator` user.

**Inputs:** Accepts the following arguments:

- `userEntitlement`: View of a `UserEntitlement` object
- `lhcontext`: `LighthouseContext` of the caller
- `objectowners`: List of Identity Manager user names that are considered owners
- `objectapprovers`: List of Identity Manager user names that are considered approvers

You must specify the following for a custom Object Approvers As Attestors rule:

**AuthType:** `AccessScanRule`

**SubType:** `ATTESTORS_RULE`

**Called:** By a running Access Review

**Returns:**

**Predefined Rule:** None

## Object Owners As Attestors

The Object Approvers As Attestors rule returns the `objectowners` parameter if it is not null. If the `objectowners` list is not provided, the rule creates a new list and includes the `Configurator` user.

**Inputs:** Accepts the following arguments:

- `userEntitlement`: View of a `UserEntitlement` object
- `lhcontext`: `LighthouseContext` of the caller
- `objectowners`: List of Identity Manager user names that are considered owners
- `objectapprovers`: List of Identity Manager user names that are considered approvers

You must specify the following for a custom Object Approvers As Attestors rule:

**AuthType:** `AccessScanRule`

**SubType:** `ATTESTORS_RULE`

**Called:** By a running Access Review

**Returns:** A list of Identity Manager user names

**Predefined Rule:** None

## Organization Names

The Organization Names rule returns a List of Display Names for all organizations within the current context.

**Inputs:** None

You must specify the following for a custom Organization Names rule:

**AuthType:** Not specified

**SubType:** Not specified

**Called:**

**Returns:**

**Predefined Rule:** None

## OS400UserFormRules

Use the OS400UserFormRules to manage the default User Form values for an OS400 resource.

**Inputs:** None

You must specify the following for a custom OS400UserFormRules rule:

**AuthType:** `EndUserLibrary`

**SubType:** Not specified

**Called:**

**Returns:** See Table 1-9

**Predefined Rule:** OS400 User Form Default Values

The following table lists the example OS400UserFormRules.

**Table 1-9** Example OS400UserFormRules

| Rule Name | Description |
|---|---|
| Default Password Expiration Interval | Returns the default value for the password expiration interval.<br>The returned value is 90. |
| Default Initial Program Call | Returns the default initial program called for a user.<br>The returned value is `*LIB/CCTC00CLP`. |
| Max Storage List Choices | Returns a list of Max Storage Defaults. The values are in Kilobytes and equate to: No maximum, 10MB, 50MB, 100MB. |
| Initial Menu Default | Returns the initial menu default value.<br>The returned value is `*SIGNOFF`. |
| Language ID Default | Returns the default language ID value.<br>The returned value is `*SYSVAL`. |
| Country ID Default | Returns the default country ID value.<br>The returned value is `*SYSVAL`. |
| Character Set Default | Returns a list of the default character set values.<br>The returned value is `*SYSVAL`. |
| UID Default | Returns the UID default value.<br>The returned value is `*GEN`. |
| Home Directory Prepend | Path to prepend to user ID to form home directory. |

## RACFUserFormRules

Use the RACFUserFormRules to specify default settings for your RACF resource account.

**Inputs:** None

You must specify the following for a custom RACFUserFormRules rule:

**AuthType:** `EndUserLibrary`

**SubType:** Not specified

**Called:** From RACF User Form

**Returns:** A list of zero or more string values.

**Predefined Rule:** RACF User Form Default Values

The following table lists the example RACFUserFormRules.

**Table 1-10** Example RACFUserFormRules

| Rule Name | Description |
|---|---|
| Prepend RACF Home Dir Path | Path prepended to `accountId` to form home directory. |
| RACF OMVS Program | Specify a default OMVS program value. |
| RACF TSO Command | Specify a default OMVS TSO value. |
| RACF Master Catalog | Specify a default OMVS program value. |
| RACF User Catalog | Specify a default OMVS program value. |
| RACF Delete TSO Segment | Specify a default Delete TSO Segment value. |

## Reconciliation Rules

The following table provides information about the common Identity Manager processes or tasks related to the reconciliation rules category:

- Correlation Rule
- Confirmation Rule

### *Correlation Rule*

Identity Manager invokes the Correlation rule during reconciliation to associate a resource account with one or more Identity Manager users.

**Inputs:** Accepts a `WSUser` representing a resource account as returned by `ResourceAdapter#getUser(WSUser)`

You must specify the following for a custom Correlation rule:

**AuthType:** Not specified

**SubType:** `SUBTYPE_ACCOUNT_CORRELATION_RULE`

**Namespace:** All attribute values for the resource account defined in the schema are provided in the following format:

`account.` *`LHS Attr Name`*

**Called:** During reconciliation

**Returns:** Criteria you can use to select existing users that might own the specified account. A correlation rule can return criteria in any of the following forms:

- A string that is interpreted as a `WSUser NAME`
- A list of string elements that are each interpreted as a `WSUser NAME`
- A list of `com.waveset.object.WSAttribute` elements
- A list of `com.waveset.object.AttributeCondition` elements

Identity Manager uses any set of criteria returned by a correlation rule to query the repository for matching users.

**Predefined Rule:** Default Correlation

### *Confirmation Rule*

Identity Manager invokes the Confirmation rule during reconciliation to compare a resource account with one or more Identity Manager users.

**Inputs:** Accepts the following arguments:

- A `WSUser` representing an existing IDM user
- A `WSUser` representing a resource account as returned by `ResourceAdapter#getUser(WSUser)`

You must specify the following for a custom Confirmation rule:

**AuthType:** None

**SubType:** `SUBTYPE_ACCOUNT_CONFIRMATION_RULE`

**Namespace:** All attribute values for the resource account and all attributes in the User view are provided in the following format:

- `account.`*LHS Attr Name*
- `user.accounts[*].*`
- `user.waveset.*`
- `user.accountInfo.*`

**Called:** During reconciliation

**Returns:** Logical true or false (`1` or `0`) depending on whether there is a match.

**Predefined Rule:** Default Confirmation

## RegionalConstants Library

The RegionalConstants Library is a default library of rules that enable you to control how states, days, months, countries, and provinces are displayed.

**NOTE** This library is displayed as the `RegionalConstants Rules` library object in the Identity Manager IDE.

**Inputs:** See Table 1-11.

You must specify the following for a custom RegionalConstants Library rule:

**AuthType:** `EndUserRule`

**SubType:** Not specified

**Called:**

**Returns:** A list of strings.

**Predefined Rule:** Regional Constants

The following table lists the example RegionalConstants rules.

**Table 1-11** Example Regional Constants Rules

| Rule Name | Input Variable | Description |
|---|---|---|
| US States | None | Returns a list of the US state names. |
| US State Abbreviations | None | Returns a list of the standard US state abbreviations. |
| Days of the Week | None | Returns a list of the full names of the seven days of the week. |
| Work Days | None | Returns a list of the five work days of the week (U.S.). |
| Months of the Year | None | Returns a list of the full names of the months of the year. |
| Month Abbreviations | None | Returns a list of the standard abbreviation for the selected month. |
| Numeric Months of the Year | None | Returns a list of 12 months. |
| Days of the Month | None | Returns a list of 31 days. |

**Table 1-11** Example Regional Constants Rules *(Continued)*

| Rule Name | Input Variable | Description |
|---|---|---|
| Smart Days of the Month | • `month`: Month whose dates are to be calculated.<br>• `year`: Year for the month whose dates are to be calculated. | Returns a list based on a numeric month and four-digit year. |
| Countries | None | Lists the names, in English, of the countries of the world. |
| Canadian Provinces | None | Lists the names, in English, of the Canadian provinces. |

## Remediation Transaction Message

The Remediation Transaction Message rule is a default rule that is used to format the remediation or mitigation transaction text. You can customize this rule to provide more information for the user to sign.

**Inputs:** Accepts the following arguments:

- `workItemList`: A set of `workitems` that are being approved.
- `variablesList`: A set of variables corresponding to each `workitem` in `workitemList`.
- `approverName`: User being asked to approve the `workitems`.
- `action`: Expected to be `remediate` or `mitigate`.
- `Comments`: Comments that are entered as part of the remediation.
- `expiration`: ISO date string for the remediation end date, which is needed only if the action is `mitigate`.

You must specify the following for a custom Remediation Transaction Message rule:

**AuthType:** `EndUserAuditorRule`

**SubType**: Not specified

**Called:**

**Returns:** Formatted remediation or mitigation transaction text

**Predefined Rule:** None

## Remediation Transaction Message Helper

The Remediation Transaction Message Helper rule returns the formatted transaction text for the remediation or mitigation of a single `workitem`.

**Inputs:** Accepts the following arguments:

- `workItem`: The `workitem` that is being approved.
- `variables`: The `workitem` variables.

You must specify the following for a custom Remediation Transaction Message Helper rule:

**AuthType:** `EndUserAuditorRule`

**SubType:** Not specified

**Returns:** Formatted remediation or mitigation transaction text

**Predefined Rule:** None

## ResourceFormRules

The ResourceFormRules library is a default library of rules that enable you to customize values and choices used in several of the UserForms, which in turn are frequently used to select user attributes for resources.

**Inputs:** See Table 1-12.

You must specify the following for a custom ResourceFormRules rule:

**AuthType:** `EndUserLibrary`

**SubType:** Not specified

**Called:** By UserForms, specifically

- `sample\forms\AccessEnforcerUserForm.xml`
- `sample\forms\ADUserForm.xml`
- `sample\forms\AIXUserForm.xml`
- `sample\forms\HP-UXUserForm.xml`
- `sample\forms\NDSUserForm.xml`
- `sample\forms\RedHatLinuxUserForm.xml`

- `sample\forms\SolarisUserForm.xml`
- `sample\forms\SUSELinuxUserForm.xml`

**Returns:** A list of strings

**Predefined Rule:** ResourceFormRuleLibrary

The following table describes the example ResourceFormRules.

**Table 1-12** Example ResourceFormRules

| Rule Name | Input Variable | Description |
|---|---|---|
| ListObjects | • `resourceType`<br>• `resourceName`<br>• `resourceInstance` | Returns a list of resource objects, such as `groups`, that can be used by multiple forms. |
| ListGroups | • `resourceName`<br>• `resourceInstance` | Returns a list of groups that can be used by multiple forms. **NOTE**: This rule is provided for backward compatibility. |
| getDefaultShell | `resourceType` | Returns a the default shell for a particular resourceType that can be used by multiple forms. Ensure that each resourceType has the same default shell as specified in the ResourceAdapter. |
| Exchange Servers | None | Returns a list of Exchange servers.<br>You can update this list to include the Exchange servers in your environment. |
| Home Directory Servers | None | Returns a list of systems serving user home directories.<br>You can update this list to include the systems that serve home directory drives in your environment. |
| AD Login Scripts | None | Returns a list of user login scripts.<br>You can update this list to include the login batch scripts in your environment. |
| Home Directory Drive Letters | None | Returns a list of home directory mapped drive letters.<br>You can update this list to include the common home directory map drive letters in your environment. |
| Home Directory Volumes | None | Returns a list of home directory volume names.<br>You can update this list to include the common home directory volume names in your environment. Identity Manager uses this value with the Home Directory server to create a user's home directory. The volume must exist and it must be shared on the selected home directory server. |

**Table 1-12** Example ResourceFormRules

| Rule Name | Input Variable | Description |
|---|---|---|
| NDS Home Directory Servers | None | Returns a list of systems serving user home directories.<br>You can update this list to include the systems that serve home directory drives in your environment. |
| NDS Home Directory Types | None | Returns a list of home directory mapped drive letters.<br>You can update this list to include the common home directory map drive letters in your environment. |
| NDS Home Directory Volumes | None | Returns a list of home directory volume names.<br>You can update this list to include the common home directory volume names in your environment. Identity Manager uses this value with the Home Directory server to create a user's home directory. The volume must exist and it must be shared on the selected home directory server. |
| NDS Template | • `resourceName`<br>• `ndsTemplate`<br>• `attrList` | Returns an NDS Template object. |
| Is Mail User | `objectClasses` | Returns `1` if the `objectClasses` list contains all the following classes, otherwise returns `0`:<br>• `inetuser`<br>• `ipuser`<br>• `inetmailuser`<br>• `inetlocalmailrecipient`<br>• `userpresenceprofile` |
| getResourceAttribute | • `resName`<br>• `attrNam` | Returns the value of the requested resource attribute. |

## Resource Names

The Resource Name rule returns a list of Resources within the current context.

**Inputs:** None

You must specify the following for a custom Resource Names rule:

**AuthType:** Not specified

**SubType:** Not specified

**Called:**

**Returns:** A list of Resources

**Predefined Rule:** None

## Role Approvers

The Role Approvers rule provides a list of users who are approvers for a specified role.

**Inputs:** Accepts the `roleName` argument

You must specify the following for a custom Role Approvers rule:

**AuthType:** `RoleUserRule`

**SubType:** Not specified

**Called:**

**Returns:** A list of the statically defined approvers for a given role

**Predefined Rule:** None

## Role Notifications

The Role Notifications rule provides a list of users who are designated to be notified when a role is assigned to a user.

**Inputs:** Accepts the `roleName` argument

You must specify the following for a custom Role Notifications rule:

**AuthType:** `RoleUserRule`

**SubType:** Not specified

**Called:**

**Returns:** A list of the statically defined administrators to notify for a given role

**Predefined Rule:** None

## Role Owners

The Role Owners rule provides a list of users who are the owners of a specified role.

**Inputs:** Accepts the `roleName` argument

You must specify the following for a custom Role Owners rule:

**AuthType:** `RoleUserRule`

**SubType:** Not specified

**Called:**

**Returns:** A list of the statically defined owners for a given role

**Predefined Rule:** None

## Sample On Local Network

The Sample On Local Network rule is an example of a LoginConstraintRule evaluated during login to determine if the login module group will be applied to the user login.

**Inputs:** None

You must specify the following for a custom Sample On Local Network rule:

**AuthType:** `LoginConstraintRule`

**SubType:** Not specified

**Called:** During login processing by the login module group

**Returns:**

- Returns **`1`** (true) if the user IP address matches a specific subnet so the login module group should be applied.
- Returns **`0`** (false) if the user IP address does not match a specific subnet.

**Predefined Rule:** None

## SAP Portal User Form Default Values

The SAP Portal User Form Default Values library is a default library of rules that provide default values for the SAP Portal User Form.

**Inputs:** None

You must specify the following for a custom SAP Portal User Form Default Values rule:

**AuthType:** `Library`

**SubType:** Not specified

**Returns:** See Table 1-13.

**Predefined Rule:** None

The following table describes the example SAP Portal User Form Default Values.

**Table 1-13** Example SAP Portal User Form Default Values Rules

| Rule Name | Input Variable | Description |
|---|---|---|
| Countries-ISO3166 Map | None | Returns a map of ISO3166 country codes. |
| Currency Code Map | None | Returns a map of country codes. |
| Locale Map | None | Returns a map of locales. |
| TimeZones | None | Returns a list of timezone IDs. |

## ShellRules

The ShellRules library consists of one rule, called `getDefaultShell`. Multiple forms use the `getDefaultShell` rule to return the default shell for a particular Unix `resourceType`.

**Inputs:** Accepts the `resourceType` argument.

The only valid `resourceTypes` are Solaris, AIX, HP-UX, and Red Hat Linux

**NOTE** Each `resourceType` must have the same default shell as specified in the ResourceAdapter.

You must specify the following for a custom ShellRules rule:

**AuthType:** Not specified

**SubType:** Not specified

**Returns:** A string that contains the default shell for the specified `resourceType`.

**Predefined Rule:** None

## SIEBEL_NAV_RULE

The SIEBEL_NAV_RULE is a sample navigation rule that could be specified as the *AdvancedNavRule*, as discussed in the "Advanced Navigation" section of the Siebel CRM documentation.

**Inputs:** None

You must specify the following for a custom SiebelNavigationRule:

**AuthType:** Not specified

**SubType:** Not specified

**Predefined Rule:** None

## TestDictionary

Use the TestDictionary rule to run a JDBC query against the Identity Manager dictionary to test the connection.

**Inputs:** Accepts the following arguments:

- `type`
- `driverClass`
- `driverPrefix`
- `url`
- `host`
- `port`
- `database`
- `context`
- `user`
- `password`
- `sql`
- `arg1`

You must specify the following for a custom TestDictionary rule:

**AuthType:** Not Specified

**SubType:** Not Specified

**Called:**

**Returns:**

**Predefined Rule:** None

## TopSecretUserFormRules

Use the TopSecretUserFormRules to specify default settings for your TopSecret resource account.

**Inputs:** None

You must specify the following for a custom TopSecretUserFormRules rule:

**AuthType:** `EndUserLibrary`

**SubType:** Not specified

**Called:** From TopSecret User Form

**Returns:** See Table 1-14.

**Predefined Rule:** None

The following table describes the example TopSecretUserFormRules.

**Table 1-14** Example TopSecretUserFormRules

| Rule Name | Description |
|---|---|
| TopSecret Default OMVS | Determines the default OMVS shell. |
| TopSecret Default TSO | Determines the default TSO Process. |
| TopSecret Home Prepend Path | Path to prepend to `accountId` to create home directory. |
| TopSecret Attribute List | Returns a list of attributes that can be assigned to a user. |

## User Members Rule

The User Members Rule enables you to dynamically control a single organization's user membership, based on who is logged in. For example, if you assign the User Members Rule to the `My Employees` organization, the rule dynamically controls the organization's user membership as follows:

- If Bob logs in and controls the `My Employees` organization, then Bob can only see and manage his employees in the `My Employees` organization.
- If Mary logs in and also controls the `My Employees` organization, she can only see and manage her employees. She cannot see or manage Bob's or anyone else's employees.

**Inputs:** User view of the authenticated admin user, context or Identity Manager session of authenticated administrator user

You must specify the following for a custom User Members Rule rule:

**AuthType:** `UserMembersRule`

**SubType:** Not specified

**Called:**

**Returns:**

- A list of resource accountIds

  You can return resource accountids by invoking the `FormUtil.getResourceObjects` call to, for example, return all user entries in a specified directory OU.

  Returned resource accountIds must be in one of the following formats:

  - `resourceId:accountId`
  - `resourceId@accountId`

```
<list>
   <s>res1:stevel</s>
   <s>res1:joem</s>
   <s>res1:sallyp</s>
 </list>
```

- A list of Identity Manager `AttributeConditions` used to query the Identity Manager repository for users matching the specified condition.

```
<list>
    <new class='com.waveset.object.AttributeCondition>
      <s>idmManager</s>
      <s>equals</s>
      <ref>waveset.accountId</s>
    </new>
  </list>
```

**Predefined Rule:** None

## USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CONF

The USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CONF rule is a confirmation rule that compares an Identity Manager user to an account.

**Inputs:** None

You must specify the following for a custom USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CONF rule:

**AuthType:** Not specified

**SubType:** `SUBTYPE_ACCOUNT_CONFIRMATION_RULE`

**Returns:** `True` if the `email` attribute values match

**Predefined Rule:** None

## USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR

The USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR rule is a correlation rule that searches for a Identity Manager user with an `email` attribute value that matches the `email` attribute value in the specified account.

**Inputs:** None

You must specify the following for a custom USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR rule:

**AuthType:** Not specified

**SubType:** `SUBTYPE_ACCOUNT_CORRELATION_RULE`

**Returns:** A list of attribute conditions

**Predefined Rule:** None

## USER_FIRST_AND_LAST_NAMES_MATCH_ACCOUNT

The USER_FIRST_AND_LAST_NAMES_MATCH_ACCOUNT rule is a confirmation rule that compares an Identity Manager user to an account by looking for a `fullname` attribute.

**Inputs:** None

You must specify the following for a custom USER_FIRST_AND_LAST_NAMES_MATCH_ACCOUNT rule:

**AuthType:** Not specified

**SubType:** `SUBTYPE_ACCOUNT_CONFIRMATION_RULE`

**Returns:** `True` if `first name` and `last name` values match, otherwise returns `false`

**Predefined Rule:** None

## USER_NAME_MATCHES_ACCOUNT_ID

The USER_NAME_MATCHES_ACCOUNT_ID rule is a correlation rule that searches for an Identity Manager user with the same name as the user in the specified account.

**Inputs:** None

You must specify the following for a custom USER_NAME_MATCHES_ACCOUNT_ID rule:

**AuthType:** Not specified

**SubType:** `SUBTYPE_ACCOUNT_CORRELATION_RULE`

**Returns:** Returns a string value

**Predefined Rule:** None

## USER_OWNS_MATCHING_ACCOUNT_ID

The USER_OWNS_MATCHING_ACCOUNT_ID rule is a correlation rule that searches for any Identity Manager user that owns an `accountId` matching the name of the specified account.

**Inputs:** None

You must specify the following for a custom USER_OWNS_MATCHING_ACCOUNT_ID rule:

**AuthType:** Not specified

**SubType:** `SUBTYPE_ACCOUNT_CORRELATION_RULE`

**Returns:** Returns a list of attribute conditions

**Predefined Rule:**

## Users Without a Manager

The Users Without a Manager rule determines which Identity Manager users are administrators.

**Inputs:** None

**NOTE** This rule uses the `lhcontext` variable from the calling scope.

You must specify the following for a custom Users Without a Manager rule:

**AuthType:** `AccessScanRule`

**SubType:** `USER_SCOPE_RULE`

**Called:**

**Returns:** A list of user names that do not have a manager defined.

**Predefined Rule:** None

## Use SubjectDN Common Name

The Use SubjectDN Common Name rule to return a subject's common name from the subject's DN.

**Inputs:** None

You must specify the following for a custom Use SubjectDN Common Name rule:

**AuthType:** `NewUserNameRule`

**SubType:** Not specified

**Called:**

**Returns:** A common name

**Predefined Rule:** None

# 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 1-15 provides an overview of the rules you can use to customize how audit policy remediation works and how access scans operate.

**Table 1-15** Auditor Rule Types Quick Reference

| Rule Type | Example Rules | subTypes and authTypes | Purpose |
|---|---|---|---|
| Attestor | Default Attestor | **SubType**: `ATTESTORS_RULE`<br>**AuthType**: `AccessScanRule` | Automates the attestation process by specifying a default attestor for manual entitlements. |
| Attestor Escalation | Default EscalationAttestor | **SubType**: `AttestorEscalationRule`<br>**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`<br>**SubType**: `SUBTYPE_AUDIT_POLICY_SOD_RULE`<br>**AuthType**: `AuditPolicyRule` | Compares user accounts to accounts specified by current Roles. |
| | Compare Roles to Actual Resource Values | **SubType**: `SUBTYPE_AUDIT_POLICY_RULE`<br>**SubType**: `SUBTYPE_AUDIT_POLICY_SOD_RULE`<br>**AuthType**: `AuditPolicyRule` | Compares current resource attributes with those specified by current Roles. |
| Remediation User Form | | **SubType**: `USER_FORM_RULE`<br>**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`<br>**AuthType**: `AccessScanRule` | Automates the remediation process by specifying a remediator for any entitlements created in remediating state. |

**Table 1-15** Auditor Rule Types Quick Reference*(Continued)*

| Rule Type | Example Rules | subTypes and authTypes | Purpose |
|---|---|---|---|
| Review Determination | Reject Changed User | **SubType**: REVIEW_REQUIRED_RULE<br>**AuthType**: AccessScanRule | Automates the attestation process by automatically rejecting user entitlement records. |
| | Review Changed Users | **SubType**: REVIEW_REQUIRED_RULE<br>**AuthType**: AccessScanRule | Automates the attestation process by automatically approving user entitlement records. |
| | Review Everyone | **SubType**: REVIEW_REQUIRED_RULE<br>**AuthType**: AccessScanRule | Automates the attestation process by requiring manual attestation for some user entitlement records. |
| User Scope | All Administrators | **SubType**: USER_SCOPE_RULE<br>**AuthType**: AccessScanRule | Provides flexibility in selecting a list of users to be scanned by an access scan. |
| | All Non-Administrators | **SubType**: USER_SCOPE_RULE<br>**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<br>**AuthType**: AccessScanRule | Provides flexibility in selecting a list of users to be scanned by an access scan. |
| ViolationPriority | ViolationPriority | **SubType**: Not specified<br>**AuthType**: EndUserAuditorRule | Customization — allows the deployment to specify what are valid violation priorities and the corresponding display strings. |
| ViolationSeverity | ViolationSeverity | **SubType**: Not specified<br>**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

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 arguments:

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

- `objectowners`:
- `objectapprovers`:

You must specify the following for a custom Attestor rule:

**AuthType:** `AccessScanRule`

**SubType:** `ATTESTORS_RULE`

**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** 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 arguments:

- `wfcontext`: WorkflowContext
- `userEntitlement`: Current 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:

**AuthType:** `AccessScanRule`

**SubType:** `AttestorEscalationRule`

**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:

**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.

**SubTypes:**

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

  SOD (*separation of duties* 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.

**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 each rule that returns a non-zero. Setting this element to zero allows the rule to return `true`, but does not create a violation for the rule.

```
<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.

**NOTE** The `RULE_EVAL_COUNT` value equals the number of rules that were evaluated during a policy scan. Identity Manager calculates this value as follows:

```
RULE_EVAL_COUNT = # of users scanned x (# of rules in policy + 1)
```

The `+1` is included in the calculation because Identity Manager also counts the *policy rule*, which is the rule that actually decides if a policy is violated. The policy rule inspects the audit rule results, and performs the boolean logic to come up with a policy result.

For example, if you have Policy A with three rules and Policy B with two rules, and you scanned ten users, the `RULE_EVAL_COUNT` value equals 70 because

```
10 users x (3 + 1 + 2 + 1 rules)
```

## 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` argument (Remediation WorkItem)

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

**AuthType:** Not specified

**Subtype:** `USER_FORM_RULE`

**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 Rule:** 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 arguments:

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

You must specify the following for a custom Remediator rule:

**AuthType:** `AccessScanRule`

**SubType:** `REMEDIATORS_RULE`

**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** 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 the record must be manually attested. A *user entitlement* is a complete User view (in which some resources might be omitted) and some tracking data.

You can use the Review Determination rule to significantly increase the efficiency of an access review by

- Encapsulating any institutional knowledge that would allow a user to be automatically approved or rejected. If you express that knowledge in this rule, you reduce the number of manual attestations needed and improve overall review performance.
- Configuring this rule to return information that is visible to the attestors as a "hint." For example, when the rule determines that a user has privilege access to a resource, the rule provides a hint to the attestor, as shown in the following example:

```
<map>
   <result>
   <i>1</i>
   <s>reason</s>
   <s><reason the attestation was auto-approved/rejected></s>
    <s>attestorHint</s>
    <s><hint to attestor></s>
</map>
```

- Configuring the rule to access the User view (including any Compliance Violations) and compare the user's previous user entitlements, which allows the rule to approve or reject all user entitlements that are the same as (or different from) a previously approved user entitlement.

You can add an argument that allows the rule to compare subsets of the User view. For example:

```
<set name='viewCompare'>
<!-- compare the entire view (3rd argument can specify sub-path) -->
<invoke name='compareUserViews' class='com.sun.idm.auditor.ui.FormUtil'>
<ref>userView</ref>
<ref>lastUserView</ref>
<s>accounts</s>
</invoke>
</set>
```

This argument compares User views and allows the caller to specify a subpath of the complete User view using `GenericObject` path expressions. If you just want to compare particular account data, the subpath can specify that data. If you compare just the `accounts` subpath of the User view, you are less likely to encounter differences that are not reflected on a real resource.

Differences found in the User view comparison are returned in the `reason` element of the output map. The audit log captures this `difference` data if the rule returns `0` (reject attestation) or `2` (approve attestation), just as the predefined Reject Changed Users rule does.

You can use the Reject Changed Users rule to verify exactly what Identity Manager thinks is different and you can look at the auditable attributes in the resulting audit log records.

**Inputs:** Accepts the following arguments:

- `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:

**AuthType:** `AccessScanRule`

**SubType:** `REVIEW_REQUIRED_RULE`

**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 user entitlement, and 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 user entitlement. The rejection comment indicates that group membership is disallowed.

```
<map>
  <s>result</s>
  <i>0</i>
  <s>reason</s>
  <s>User belongs to 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. The rule only compares the `accounts` section of the User view.

  Each unknown User view is forwarded for manual attestation.

- **Review Changed Users**: Automatically approves any users whose account data has not changed since their last approved entitlement. The rule only compares the `accounts` section of the User view.

  Users with changed account data or no approved data must be manually attested.

- **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` argument

You must specify the following for a custom User Scope rule:

**AuthType:** `AccessScanRule`

**SubType:** `USER_SCOPE_RULE`

**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:

**AuthType:** `EndUserAuditorRule`

**SubType:** Not specified

**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:

**AuthType:** `EndUserAuditorRule`

**SubType:** Not specified

**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 1-31).

**Code Example 1-31** 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`

# Audit Policy 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 the seriousness of the violation.
- **ViolationPriority**: Indicates the order in which a ComplianceViolation would be addressed.

Identity Auditor allows you to customize these rules by changing the display value for any severity or priority setting.

After creating a ComplianceViolation, you can view and change the 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 a more-detailed view affects performance, which may be unacceptable for deployments with lots of Remediations.

# Service Provider Rules

This section describes the following example Service Provider rules:

- Service Provider Confirmation Rules
- Service Provider Correlation Rules
- Service Provider Account Locking Rules

## Service Provider Confirmation Rules

The example Service Provider confirmation rules have access to the list of candidate accountIds under the *candidates* path and to the Service Provider User view under the *view* path.

**Inputs:** None

You must specify the following for a custom Service Provider confirmation rule:

**AuthType:** `SPERule`

**SubType:** `SUBTYPE_SPE_LINK_CONFIRMATION_RULE`

**Returns:** A null or a string representing the confirmed accountId

**Predefined Rule:** None

The following table describes the example confirmation rules you can use to customize Service Provider.

**Table 1-16** Example Service Provider Confirmation Rules

| Rule Name | Description |
|---|---|
| Service Provider Example Confirmation Rule Rejecting All Candidates | Rejects all candidates from a link correlation rule. Returns a null. |
| Service Provider Example Confirmation Rule Returning First Candidate | Returns the first accountId from the candidate list. |
| Service Provider Example Confirmation Rule Selecting Candidates Using AccountId | Returns the candidate that matches the accountId in the view. If the rule cannot find the accountId from the view in the candidate list, then the rule returns a `null`. |

## Service Provider Correlation Rules

The example Service Provider correlation rules have access to the Service Provider User view.

**Inputs:** None

You must specify the following for a custom Service Provider correlation rule:

**AuthType:** `SPERule`

**SubType:** `SUBTYPE_SPE_LINK_CORRELATION_RULE`

**Returns:** A single accountId, a list of accountIds, or an option map

- If the rule returns a list of accountIds, then you must set a confirmation rule to determine the selected accountId.
- If the rule returns an option map, then the view handler first retrieves a list of identities from the resource adapter by invoking the `listResourceObjects` context call with the provided option map.

**Predefined Rule:** None

The following table describes the example correlation rules you can use to customize Service Provider.

**Table 1-17** Example Service Provider Correlation Rules

| Rule Name | Description |
|---|---|
| Service Provider Example Correlation Rule for LDAP Returning Option Map | Returns an option map with a search filter to be used with an LDAP adapter. The LDAP Resource Adapter allows a filter to be passed to scope the search operation. The filter is expected to be an LDAP search filter. |
| Service Provider Example Correlation Rule for Simulated Returning Option Map | Returns an option map with a search filter to be used with a Simulated Resource Adapter. The Simulated Resource Adapter allows a filter to be passed to scope the search operation. This adapter expects the search filter to be an `AttributeExpression`. |
| Service Provider Example Correlation Rule Returning List of Identities | Returns a list of accountIds in LDAP DN format that are composed from the accountId in the view. |
| Service Provider Example Correlation Rule Returning Single Identity | Returns a single accountId in LDAP DN format composed from the account Id in the view. |

## Service Provider Account Locking Rules

The example Service Provider account locking rules have access to the Service Provider User view and they lock or unlock accounts in a Sun Directory Server.

**Inputs:** See Table 1-18.

You must specify the following for a custom Service Provider account locking rule:

**AuthType:** `SPERule`

**SubType:** Not specified

**Returns:** Nothing

**Predefined Rule:** None

The following table describes the example account locking rules you can use to customize Service Provider.

**Table 1-18** Example Service Provider Account Locking Rules

| Rule Name | Input Variable | Description |
|---|---|---|
| Service Provider Example Lock Account Rule | `lockExpirationDate`: A possibly null `java.util.Date` at which the lock should expire. | Locks an account in a Sun Directory Server. This rule modifies top-level attributes in the Service Provider user view. |
| Service Provider Example Unlock Account Rule | None | Unlocks an account in a Sun Directory Server. This rule modifies top-level attributes in the Service Provider user view. |

Chapter 2

# Developing Custom Adapters

Identity Manager's open architecture enables you to create custom resource adapters to manage external resources that are not already supported by the resource adapters provided with Identity Manager. These custom adapters define essential characteristics and methods needed to transform requests from Identity Manager into actions performed on a resource.

This chapter describes how to create, test, and load a custom Identity Manager resource adapter. This information is organized as follows:



**NOTE** Identity Manager contains sample adapters or *skeleton adapters* that are intended to be used as a basis for creating custom adapters. To enhance your understanding of these valuable starter files, this chapter uses them frequently to exemplify particular characteristics of resource adapter files.

# Before You Begin

Review the information in these sections before you start developing custom adapters:



## Intended Audience

This chapter provides background information about resource adapter design and operation for:

- Developers who need to create custom resource adapters
- Identity Manager administrators who are learning how the Identity Manager system works or who need to troubleshoot problems with resource adapters.

This chapter assumes that you are familiar with creating and using the built-in Identity Manager resources and that you have read the Resources chapter of *Sun Java™ System Identity Manager Administration*.

## Important Notes

Be sure to read the following information before trying to write custom resource adapters for Identity Manager:

- Do not create custom adapters in the `com.waveset.adapter` package. Instead, create custom adapters in a customer-specific package to be sure the adapter uses package-level classes and methods that are included in the supported public API. For example, use com.*customer_name*.adapter.

  Also, all package names must be lowercase.

- Do not use `import .*`. Although Java supports this mechanism, using `import .*` is generally considered bad practice because this mechanism
  - Obscures the actual location of referenced classes to readers
  - Can result in incorrect or ambiguous references (such as compiler errors) in certain situations following internal refactoring

  Instead, insert an explicit `import` statement for every referenced class or interface.

## Related Documentation

In addition to the information provided in this chapter, see the following publications related to resource adapters:

**Table 2-1** Related Documentation

| Publication Title | Description |
|---|---|
| *Identity Manager Resource Reference* | Describes how to load and synchronize account information from a resource into Sun™ Identity Manager. |
| *Identity Manager Administration* | Contains additional information about customizing and managing resources supplied by Identity Manager. |

You can download these publications from `http://docs.sun.com`.

# What is a Resource Adapter?

A resource adapter serves as a proxy between Identity Manager and an external resource, such as an application or database. The adapter defines the essential characteristics of the resource type, and this information is saved in the Identity Manager repository as a *resource object*. Identity Manager resource adapters are *standard* or *Active Sync-enabled* adapters.

This section contains the following topics:

# What Are Standard Resource Adapters?

Standard resource adapters provide a generic interface to resource types that are supported by Identity Manager; such as Web servers, Web applications, databases, and even legacy applications and operating systems. In Java terms, standard resource adapters extend the `ResourceAdapterBase` class.

These adapters push account information changes from Identity Manager to their managed, external resources and typically perform the following administrative activities:

- Connect to and disconnect from a resource
- Create, delete, or modify users
- Enable, disable, or get users
- Authenticate users
- Manage objects such as group membership or directory organization structure

Standard resource adapters generally follow these 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 that creates the form to display the resource type's prototype definition is queried for the resource attribute fields. Identity Manager uses these attributes to display a form in the browser. The user who is creating the new resource fills in the information and clicks Save.

3. Identity Manager saves the information provided, 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 request is a create request, the adapter updates the Identity Manager user object 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.

# What Are Active Sync-Enabled Resource Adapters?

Active Sync-enabled adapters are an extension of a standard resource adapter and they are used to implement the Active Sync interface for some common resources, such as Active Directory. These adapters pull data changes directly from the resource to initiate the following activities in Identity Manager:

- Polling or receiving change event notification
- Issuing actions to create, update, or delete resource accounts
- Editing or creating users with a custom form
- Saving the resource changes
- Logging progress information and errors

Active Sync-enabled adapters are particularly suitable for supporting the following resource types:

- Applications with audit or notification interfaces

  Some applications, such as Microsoft Active Directory and PeopleSoft, have external interfaces. You can configure these application interfaces to add events to an audit log or to notify other applications when certain changes occur.

  For example, you can configure the interface to record an transaction in the audit log whenever a user account is modified natively on the Active Directory server. You can configure the Identity Manager Active Directory resource to review this log every 30 minutes and trigger events in Identity Manager when any changes occur. You can register other Active Sync-enabled adapters with the resource through an API, and use event messages to notify the adapter when changes occur. These event messages can reference the item that changed, the information that was updated, and frequently the user who made the change.

- Databases populated with update information

  You can manage database resources by generating a table of deltas and generate this table in several different ways. For example, you can compare a snapshot of the database to current values and create a new table with the differences. The adapter pulls rows from the table of deltas, processes them, and subsequently marks them when completed.

- Databases with modification timestamps

  You can create Active Sync-enabled queries for database entries that have been modified after a particular time. The adapters 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 the polling impact. Only those changes made to the resource since the previous set of modifications were made are returned for processing.

- Resources with change-log entries.

  Most LDAP servers provide a change-log mechanism that you can use to track changes, optionally constrained to sections of interest in the DIT. By periodically querying the change-log entries, the LDAP resource adapter can update Identity Manager with detected changes; including creates, deletes, and updates.

Active Sync-enabled adapters generally follow these 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 `IAPI` object and returns a `WavesetResult` object to the adapter. The `WavesetResult` object informs the Active Sync-enabled adapter if the operation succeeds.

   The `WavesetResult` object might contain many results from the various steps the Identity Manager system used to update the identity. Typically, a workflow also handles errors within Identity Manager, often ending up as an Approval for a managing administrator.
7. Exceptions are logged in the Active Sync and Identity Manager tracing logs with the `ActiveSyncUtil.logResourceException` method.

When Active Sync-enabled adapters detect a change to an account on a resource, the adapter maps the incoming attributes to an Identity Manager user or, if the adapter cannot match the user account, creates an Identity Manager user account.

The following rules and parameters determine what happens when a change is detected.

**Table 2-2** Active Sync-Enabled Adapter Rules and Parameters

| Parameter | Description |
|---|---|
| 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.<br><br>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.<br><br>The same confirmation rule can be used for reconciliation and Active Sync. |
| 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).<br><br>Returns one of the following types of information that can be used to correlate the entry with an existing Identity Manager account:<br>• Identity Manager user name<br>• `WSAttributes` object (used for attribute-based search)<br>• List of `AttributeCondition` or `WSAttribute`-type items (`AND`-ed attribute-based search)<br>• List of `String`-type items (each item is the Identity Manager ID or the user name of an Identity Manager account)<br><br>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.<br><br>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.<br><br>The same correlation rule can be used for reconciliation and Active Sync. |
| 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`. |
| 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.<br><br>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. |
| Populate Global | If set to `true`, populates the global namespace in addition to the ActiveSync namespace. The default value is `false`. |

**Table 2-2** Active Sync-Enabled Adapter Rules and Parameters *(Continued)*

| 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.<br>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.<br>If a Process rule is specified, the process will be run for every row regardless of any other settings on this adapter.<br>At minimum, a process rule must perform the following functions:<br>• Query for a matching User view.<br>• If the User exists, checkout the view. If not, create the User.<br>• Update or populate the view.<br>• Checkin the User view.<br>It is possible to synchronize objects other than User, such as LDAP Roles. |
| 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.<br>This rule is also needed if there were no matches and Create Unmatched Accounts was not selected.<br>This workflow can be a process that prompts an administrator for manual action. |

**NOTE** If present, a Process rule determines whether the adapter uses `IAPIProcess` or attempts to use `IAPIUser`. If the adapter cannot use `IAPIUser` because a Correlation or Confirmation rule does not uniquely identify an Identity Manager user for the event (given the other parameter settings), and a Resolve Process rule is configured, the adapter uses the Resolve Process rule to create an `IAPIProcess` event. Otherwise, the adapter reports an error condition.

`IAPIUser` checks out a view and makes this view available to the User form.

- For creates and updates, `IAPIUser` checks out the `User` view.
- For deletes, `IAPIUser` checks out the `Deprovision` view.

However, a `User` view is not checked out or available with `IAPIProcess`. Either a Process rule has been set or a Resolve Process rule is invoked.

# What is a Resource Object?

Resource objects define the capabilities and configuration of the resource you are managing in Identity Manager — including the information described in the following table.

**Table 2-3** Information Defined by Resource Objects

| Type of Information | Sample Attributes |
|---|---|
| Connection information | • Host name<br>• Administrative account name<br>• Administrative account Password |
| User attributes | • First name<br>• Last name<br>• Phone numbers |
| Identity Manager attributes | • List of approvers<br>• Password policy for the resource<br>• How many times to repeat attempts when contacting the resource |

You must define a resource object in Identity Manager for every resource that Identity Manager communicates with or manages.

**NOTE** You can view resource objects from Identity Manager's Debug pages:

`http://`*host*`:`*port*`/idm/debug/`

Where:

- *host* is the local server on which Identity Manager is running.
- *port* is the TCP port number on which the server is listening.

The `session.jsp` page gives you the option of listing objects of type `Resource`. See "Viewing and Editing a Resource Object" on page 185 for more information.

## What is a Resource Adapter Class?

A resource adapter class implements methods that

- Register the resource object in the Identity Manager repository
- Enable you to manage the external resource
- Push information from Identity Manager to the resource
- (*Optional*) Pull information from the resource into Identity Manager

  This optional pull capability is known as Active Sync, and a resource adapter with Active Sync capability is referred to as *Active Sync-enabled*. See "What Are Active Sync-Enabled Resource Adapters?" on page 105 for more information.

# Preparing for Adapter Development

Some preparation is necessary before you actually start writing a custom adapter. This section describes how to prepare for adapter development, and the tasks include:

- Become Familiar with Adapter Source Code
- Profile the Resource
- Decide Which Classes and Methods to Include
- Review the REF Kit
- Set Up the Build Environment

## Become Familiar with Adapter Source Code

Before you can create a custom adapter, you must become familiar with the components in a resource adapter's source code. This section describes the following components, which are common to most adapters:

- Standard Java Header Information
- `PrototypeXML` String
- Resource Methods

## Standard Java Header Information

Standard Java header information identifies the parent class of the new adapter class file you are creating, constructors, and imported files.

This 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` String

The `prototypeXML` string in the adapter Java file is the XML definition of a resource. This string must contain the resource name and all of the resource attributes that you want to display in the Identity Manager user interface. The `prototypeXML` string also defines resource objects stored in the Identity Manager repository.

The following table describes the different `prototypeXML` information types that you use to define a resource in Identity Manager.

**NOTE** Some of these information types are specific to Active Sync-enabled adapters.

**Table 2-4** `prototypeXML` Information Types

| Type | Description |
|---|---|
| Resource | Defines top-level characteristics of the resource.<br>Keywords include:<br>• **`syncSource`**: If `true`, then adapter must be Active Sync-enabled.<br>• **`facets`**: Specifies the modes enabled for this resource. |
| Resource attributes | XML elements that are defined with the `<ResourceAttribute>` element and used by Identity Manager to configure the resource.<br>For more information, see "Resource Attributes" on page 113. |
| Account attributes | Defines the default schema map for basic user attributes.<br>You use the `<AccountAttribute>` element to define account attributes. You map standard Identity Manager account attribute types differently than you map custom attributes.<br>For more information about mapping account attributes to resource attributes, see "Map the Attributes " on page 139. |

**Table 2-4** `prototypeXML` Information Types*(Continued)*

| Type | Description |
|---|---|
| Identity template | Defines how the account name for the user is built.<br>Use the `<Template>` tag to define this template. Account names are typically in one of the following forms:<br>• An `accountId` is typically used for resources with a flat namespace such as Oracle.<br>• A complete distinguished name (DN) of the user in the form: `cn=accountId,ou=sub-org,ou=org,o=company`. Use this form for hierarchical namespaces such as directories.<br>For more information, see "Identity Template" on page 121. |
| Login configuration | (*Standard resource adapter only*)<br>Defines values to support pass-through authentication for the resource.<br>Use the `<LoginConfigEntry>` element to define this value.<br>For more information about pass-through authentication, see the *Sun™ Identity Manager Resources Reference.* |
| Form | (*Active Sync-enabled adapters only*)<br>Designates a form object that processes data from the Active Sync-enabled adapter before the data is integrated into Identity Manager. A form is optional, but in most cases a form provides flexible changes in the future and can be used to transform incoming data, map data to other user attributes on other resource accounts, and cause other actions in Identity Manager to occur. |

### *Resource Attributes*

*Only available to Administrators defining the resource.*

Resource attributes define 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.

When writing custom adapters, you use resource attributes to 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 Active Directory 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 an Active Directory 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.

**Defining Resource Attributes.** You use the <ResourceAttribute> element, as shown in the following example, to define resource attributes in the prototypeXML string of the adapter Java file:

```
<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"+
```

Where 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 '.

The following table describes the keywords you can use in <ResourceAttribute> element.

**Table 2-5** <ResourceAttribute> Element Keywords

| Keyword | Description |
|---|---|
| name | Identifies the name of the attribute.<br>**NOTE**: The name keyword is a reserved word in views and should not be used as a Identity System User Attribute on resource schema maps. |
| type | Identifies the data type used. |
| multi | Specifies whether multiple values can be accepted for the attribute. If true, a multi-line box displays. |
| 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:<br>Description='host Enter the resource host name.' |
| facets | Specifies the usage of this resource attribute. Valid values are<br>• **provision**: Used in standard processing (default value).<br>• **activesync**: Used in Active Sync processing for an Active Sync-enabled adapter. |

You can modify these values from the Identity Manager interface when creating a specific instance of this resource type.

**Overwriting Resource Attributes.** When you are working with resource adapters and adapter parameters, you can use one of the following strategies to overwrite resource attributes:

- 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 the following example, 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 to calculate this `template` value within your user form.

**Code Example 2-1** Overwriting 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.** The following table describes required resource attributes that are supplied in the skeleton adapter files.

**Table 2-6** Resource Attributes in Skeleton Adapter Files

| Required Resource Attribute | Description |
|---|---|
| `RA_HOST` | Resource host name. This attribute corresponds to the Host field on the Resource Parameters page. |
| `RA_PORT` | Port number used to communicate with the resource. This attribute corresponds to the Port field on the Resource Parameters page. |
| `RA_USER` | Name of a user account that has permission to connect to the resource. The field name varies on the Resource Parameters page. |
| `RA_PASSWORD` | Password for the account specified by `RA_USER`. This attribute corresponds to the Host field on the Resource Parameters page. |

The next table describes required Active Sync-specific attributes that are defined in the ACTIVE_SYNC_STD_RES_ATTRS_XML string of the Active Sync class.

**Table 2-7** 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 attribute 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 attribute corresponds to the Input Form field. |
| RA_MAX_ARCHIVES | Specifies the number of log files to retain.<br>• If you specify **0**, then a single log file is re-used.<br>• If you specify **-1**, then log files are never discarded. |
| RA_MAX_AGE_LENGTH | Specifies the maximum time before a log file is archived.<br>• If you specify zero, then no time-based archival occurs.<br>• If the RA_MAX_ARCHIVES value is zero, then the active log is truncated and reused after this time period. |
| RA_MAX_AGE_UNIT | Specify seconds, minutes, hours, days, weeks, or months. Use this value with RA_MAX_AGE_LENGTH. |
| RA_LOG_LEVEL | Logging level (0 disabled; 4 very verbose). This attribute corresponds to the Log Level field in the Identity Manager display. |
| RA_LOG_PATH | Absolute or relative path for the log file. This attribute corresponds to the Log File Path field in the Identity Manager display. |
| RA_LOG_SIZE | Maximum log file size. This attribute 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 | Number of intervals between scheduled periods (for example, 10 minutes has an interval count of 10 and an interval of minute). Not necessary for Active Sync-enabled adapters. |
| RA_SCHEDULE_START_TIME | Time of the day to run. For example, if you specify 13:00 and set the interval to week, the adapter runs at 1 P.M. once a week. Not necessary for Active Sync-enabled adapters. |
| RA_SCHEDULE_START_DATE | Date to start scheduling. Setting date to 20020601, the interval to month, and the time to 13:00 starts the adapter on June 1st and runs once a month at 1 P.M. Not necessary for Active Sync-enabled adapters. |

This table describes required Active Sync-specific attributes that are defined in the ACTIVE_SYNC_EVENT_RES_ATTRS_XML string of the Active Sync class.

**Table 2-8** Active Sync-Specific Attributes Defined in ACTIVE_SYNC_EVENT_RES_ATTRS_XML

| Required Resource Attribute | Description |
|---|---|
| RA_PROCESS_RULE | Name of a TaskDefinition or a rule that returns the name of a TaskDefinition to run for every record in the feed. This attribute overrides all others. |
| RA_CORRELATION_RULE | 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 | Rule that confirms whether a user is a match. |
| RA_DELETE_RULE | 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.<br>• If false, do not create the account unless the process rule is set and the workflow it identifies determines that a create is warranted. Default is true. |
| RA_RESOLVE_PROCESS_RULE | 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. Default is false. |

### *Identity Manager Account Attributes*

*Only available to Administrators defining the resource.*

Identity Manager account attributes describe the default user attributes supported for the resource.

With an Active Sync-enabled adapter, account attributes 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.

Identity Manager supports the following types of account attributes:

- string
- integer
- boolean
- encrypted
- binary

Binary attributes include graphic files, audio files, or certificates. Not all adapters support binary account attributes. Generally, only certain directory, flat file, and database adapters can process binary attributes.

**NOTE**

- Consult the "Account Attributes" section of the adapter documentation to determine if your adapter supports binary attributes.
- Keep the size of any file referenced in a binary attribute as small as possible. For example, loading extremely large graphics files can affect Identity Manager's performance.

You define Identity Manager account attributes in the `AttributeDefinition` object of the resource's schema map, and use the `prototypeXML` string in the adapter file to map incoming resource attributes to account attributes in Identity Manager. For example, you would map the LDAP `sn` resource attribute to the `lastname` attribute in Identity Manager. Identity Manager account attributes include

- `accountId`
- `email`
- `firstname`
- `fullname`
- `lastname`
- `password`

**Standard Adapter Schema Maps.** You use the Account Attributes page, or schema map, to map Identity Manager account attributes to resource account attributes. The list of attributes varies for each resource. You generally remove all unused attributes from the schema map page. If you add attributes, you will probably need to edit user forms or other code.

The attribute mappings specified in the resource schema map determine which account attributes can be requested when you are creating a user. Based on the role selected for a user, you will be prompted for a set of account attributes that are the union of attributes of all resources in the selected role.

**NOTE** To view or edit the Identity Manager schema for users or roles, you must be a member of the `IDM Schema Configuration AdminGroup` and you must have the `IDM Schema Configuration` capability.

**Active Sync-Enabled Adapter Schema Maps.** The Active Sync resource schema map is an optional utility that enables you to edit inputs to an Active Sync-enabled adapter, which are often database column names or directory attribute names. Using the schema map and an Active Sync form, you can implement Java code to handle a resource type, defining details of the resource configuration in the map and form.

Identity Manager uses an Active Sync resource's schema map in the same way that it uses a typical schema map. The schema map 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** Do not put the `accountId` attribute in the global namespace because this special attribute is used to identify `waveset.account.global`.

If you are creating the resource account for the first time, the `accountId` attribute also becomes a resource's `accountId` directly and it bypasses 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.

**Using the Schema Map.** After creating a resource instance, administrators 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.

You can view Identity Manager account attributes from the Edit Schema page in the Identity Manager user interface by clicking the Edit Schema button located at the bottom of the Edit/Create Resource page.

For more information about creating a resource or editing a resource schema map, see *Sun Java™ System Identity Manager Administration*.

### *Identity Template*

**NOTE** An identity template is only available to Administrators who are defining the resource.

To view or edit the Identity Manager schema for Users or Roles, you must be a member of the `IDM Schema Configuration AdminGroup` and you must have the `IDM Schema Configuration` capability.

You use the identity template (or account DN) to define a user's default account name syntax when creating the account on the resource. The identity template translates the Identity Manager user account information to account information on the external resource.

You can use any schema map attribute (an attribute listed on the left side of the schema map) in the identity template, and you can overwrite the user identity template from the User form, which is commonly done to substitute organization names.

Identity Manager users have an identity for each of their accounts, and this identity can be the same for some or for all of these 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 and as a separate `accountId` for each of the resources on which that user has an account. The `accountId` is denoted in the form of `accountId:<resource name>`, as shown in the following table.

**Table 2-9** `accountID` Examples

| 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` |

Account user names are in one of two forms:

- Flat namespaces
- Hierarchical namespaces

**Flat Namespaces.** You typically use the `accountId` attribute for systems with a flat namespace, which include:

- 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 be used.

**Hierarchical Namespaces.** You use distinguished names (DNs) for systems with a hierarchical namespace. DNs can include the account name, organizational units, and organizations.

Account name syntax is especially important for hierarchical namespaces. For resources with hierarchical namespaces, the identity template can be more complicated than that of a flat namespace, which allows you to build the full, hierarchical name. The following table shows examples of hierarchical namespaces and how they represent DNs.

**Table 2-10** 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` |

For example, you can specify the following for a resource identity template 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

### *Login Configuration*

Login Configuration defines parameters that are used if you are going to use the resource for pass-through authentication. Typically, these parameters are `username` and `password`, but some resources use different parameters. For example, SecurId uses `user name` and `passcode`.

The Login Configuration information type helps define the resource, but it is not easily modified by administrators.

For more information about pass-through authentication, see "Enabling Pass-Through Authentication for Resource Types" on page 150 and the *Sun™ Identity Manager Resources Reference.*

## Resource Methods

Resource methods write information from Identity Manager into the external resource.

**NOTE** You must be familiar with the resource to write customized methods.

You categorize resource methods by task. When developing your own custom adapters, you must determine which categories your adapter needs to meet the goals of your development. For example,

- Is your adapter going to be a standard or an Active Sync-enabled adapter?
- Will the first phase of your deployment support password reset only?

How you answer these questions determines which resource methods must be completed.

The following table describes resource methods categories. (Additional information about each functional category is discussed later in this chapter.)

**Table 2-11** 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. Helps define the resource, but is not easily modified by administrators. |

In Active Sync-enabled adapters, resource methods

- Create a feed from the resource into Identity Manager. Presents methods that search the resource for changes or receive updates. To write these methods, you must understand how to register or search for changes on the resource, and communicate with the resource.
- Run update operations in the Identity Manager repository by performing the feed from the resource into Identity Manager.

## Considerations for Standard Resource Adapters

The following considerations are specific to account attributes in standard resource adapters:

- User identity template
- Creating an identity template out of multiple user attributes
- Login configuration and pass-through authentication

### *User Identity Template*

**NOTE** To view or edit the Identity Manager schema for Users or Roles you must be a member of the `IDM Schema Configuration AdminGroup` and you must have the `IDM Schema Configuration` capability.

The user identity template establishes the account name to use when creating the account on the resource. This template translates Identity Manager user account information to account information on the external resource.

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, which is commonly done to substitute organization names.

### *Creating an Identity Template Out of Multiple User Attributes*

You can create an identity template out of a portion of multiple user attributes. For example, a template might consist of the first letter of the first name plus seven characters of the last name. In this case, 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. Do not change this section of the file if you want the resource to appear in the options list.

Each <AuthnProperty> element contains the following attributes:

**Table 2-12** <AuthnProperty> Element Attributes

| Attribute | Description |
|---|---|
| dataSource | Specifies the source for the value of this property. Data sources for this property value include:<br>• **user** (*Default*): Value provided by the user at login time.<br>• **http attribute**: Value provided by the specified http session attribute.<br>• **http header**: Value provided by the specified http header.<br>• **http remote user**: Value provided by the http request's remote user property.<br>• **http request**: Value provided by the specified http request parameter.<br>• **resource attribute** (Active Directory only): Value allows you to specify an extra authentication attribute for the specific adapter. This attribute is only valid for the resource on which it is defined, and it cannot be manipulated by the user.<br>• **x509 certificate**: Value is the X509 client certificate (only valid for requests made using https). |
| displayName | Specifies the value to use when this property is added as an HTML item to the Login form. |
| doNotMap | Specifies whether to map to a LoginConfigEntry. |
| 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 if the property value is an X509 certificate. |
| name | Identifies the internal authentication property name. |

User management across forests is only possible when multiple gateways, one for each forest, are deployed. In this case, you can configure the adapters to use a predefined domain for authentication per adapter without requiring the user to specify a domain as follows:

1. Add the following authentication property to the <AuthnProperties> element in the resource object's XML:

   ```
   <AuthnProperty name='w2k_domain' dataSource='resource attribute'
   value='MyDomainName'/>
   ```

2. Replace *MyDomainName* with the domain that authenticates users.

**NOTE** For more information about this property, see the Active Directory resource adapter documentation in *Identity Manager Resources Reference*.

Most resource login modules support both the Identity Manager Administrative interface and User interface. The following example shows how SkeletonResourceAdapter.java implements the <LoginConfigEntry> element:

**Code Example 2-2** 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"+
```

The following example defines the supported LoginModule DATA_SOURCE options. In this example, a LoginConfig entry is taken from the LDAP resource adapter supplied by Identity Manager. The entry defines two authentication properties whose dataSource value, if not specified, is supplied by the user.

**Code Example 2-3** 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"+
```

The next example shows a `Login Config` entry where the authentication property's `dataSource` value is not supplied by the user. In this case, the value is derived from the HTTP request header.

**Code Example 2-4** `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"+
```

### Example Object Resource Attribute Declaration

The following example shows how prototypeXML defines fields displayed on the Create/Edit Resource page.

**Code Example 2-5** 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='AD'>
   </ResourceAttribute>
</ResourceAttributes>
```

The Identity Manager Administrative interface displays the resource attributes for the default resource as specified.

## Profile the Resource

The following sections describe how to profile and define prerequisites for standard resource adapters and Active Sync-enabled adapters.

- Profiling a Standard Resource Adapter
- Profiling an Active Sync-Enabled Resource Adapter

## Profiling a Standard Resource Adapter

Use the following information to create a profile and define prerequisites for a standard resource adapter:

- Select an Identity Manager adapter file that most closely resembles the resource type to which you are connecting.

  See Table 2-13 on page 132 for a brief description of the default Identity Manager resource adapter files supplied with a standard Identity Manager configuration.

- Research user account characteristics and how these tasks are performed on the remote resource:
  - Authenticate access to the remote resource
  - Update users
  - Get details about the changed users
  - List all users on the system
  - List other system objects, such as groups, that are used in the `listAllObjects` method
- Identify the minimum attributes needed to perform an action and all supported attributes.
- Verify that you have the appropriate tools to support connection to the resource.

  Many resources ship with a published set of APIs or a complete toolkit that can be used to integrate outside applications to the resource. Determine whether your resource has a set of APIs or whether the toolkit provides documentation and tools to speed up integration with Identity Manager. For example, you must connect to a database through JDBC.

- Determine who can log in and search for users on the resource

  Most resource adapters require and run an administrative account to perform tasks such as searching for users and retrieving attributes. This account is typically a highly privileged or super user account, but can be a delegated administration account with read-only access.

- Determine whether you can extend the resource's built-in attributes.

  For example, Active Directory and LDAP both allow you to create *extended schema* attributes, which are attributes other than the standard Identity Manager attributes.

  Decide which attributes you want to maintain in Identity Manager, determine what the attribute names are on the resource, and decide what to name the attributes in Identity Manager. These attribute names go in the schema map and are used as inputs to forms that are used to create a resource of that type.

## Profiling an Active Sync-Enabled Resource Adapter

When profiling an Active Sync-Enabled resource adapter, use the following information *in addition to* the considerations described in "Profiling a Standard Resource Adapter" on page 129:

- When researching user account characteristics and how these tasks are performed on the remote resource, you must also:
  - Search for changes to users
  - Identify ways to search for changed users only
- Determine which resource attributes or actions create events.

  If the resource supports subscribing to notification messages when changes are made, identify which attribute changes you want to trigger the notification and which attributes you want in the message.
- Decide which of the following actions Identity Manager should perform when the adapter detects an event on the source.
  - Create, update, or delete a user
  - Disable or enable an account
  - Update the answers used to authenticate a user
  - Update a phone number

- Decide whether you want the adapter to be driven by events in the external resource or driven by specified polling intervals.

  **NOTE** Before making your decision, you must understand how polling works in typical Identity Manager installations. Although some installations implement or are driven by external events, most Identity Manager deployment environments use a hybrid method.

  Choose one of the following approaches:

  - Set up polling intervals where an Active Sync Manager thread calls the poll interface 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.
  - Set up an event-driven environment where the adapter sets up a listening connection, such as an LDAP listener, and waits for messages from the remote system. You can implement the poll method to do nothing, and set the polling interval to an arbitrary value, such as once a week. If updates are event-driven, the updates must have a guaranteed delivery mechanism, such as MQ Series, or synchronization is lost.
  - Implement a hybrid solution where external events trigger *smart polling* and the regular poll routine can recover from missed messages.

    Smart polling adapts the poll rate to the change rate and polls infrequently unless changes are being made often. Smart polling balances the performance impact of frequent polling with the update delays of infrequent polling.

    In this model, incoming messages are queued and multiple updates for a single object are collapsed into a single update, which increases efficiency. For example, multiple attributes can be updated on a directory, and each attribute triggers a message. The poll routine examines the message queue and removes all duplicates. The routine then fetches the complete object to ensure that the latest data is synchronized and that updates are handled efficiently.

## Decide Which Classes and Methods to Include

After profiling the resource, identify classes and methods needed in your adapter:

- Review the relevant Javadoc to determine which base classes and methods you can use as is and which you must extend. This javadoc is available on your Identity Manager CD in the `REF/javadoc` directory:
- Create a list of methods that you must write and include in the Java file based on the resource to which you are connecting.

  When creating an adapter, the most time-intensive task is writing your own methods to push information from Identity Manager to the resource or to create a feed from the resource to Identity Manager.

## Review the REF Kit

The Sun Resource Extension Facility Kit (REF Kit) is supplied in the `/REF` directory on the Identity Manager CD or install image. You can use the sample files and other tools in this REF Kit to jump-start the process of creating your own custom adapter.

The following table describes the contents of the REF Kit.

**Table 2-13** REF Kit Components

| Component | Location | Description |
|---|---|---|
| `audit` | `REF/audit` | Sample custom audit publishers. |
| `exporter` | `REF/exporter` | Warehouse interface code source code that allows you to rebuild the warehouse interface to let Data Exporter export to something other than the warehouse relational database. |
| `javadoc` | `REF/javadoc` | Generated javadoc that describes the classes you need to write a custom adapter. To view the javadoc, point your browser to:<br>`/waveset/image/REF/javadoc/index.html` |
| `lib` | `REF/lib` | Jar files that you need to compile and test a custom adapter. |

**Table 2-13** REF Kit Components *(Continued)*

| Component | Location | Description |
|---|---|---|
| src | REF/src | Examples of commonly developed resource adapter source files and skeleton files to use as a basis for adapter development and testing, including:<br>• MySQLResourceAdapter.java for Database accounts<br>• ExampleTableResourceAdapter.java for Database tables[1]<br>• XMLResourceAdapter.java for File-based accounts<br>• LDAPResourceAdapter.java for simple methods when developing custom LDAP resource adapters<br>• LDAPResourceAdapterBase.java for complex changes when developing custom LDAP resource adapters<br>• AIXResourceAdapter.java for developing UNIX accounts<br>• SkeletonStandardResourceAdapter.java for standard resources<br>• SkeletonStandardAndActiveSyncResourceAdapter.java for standard and Active Sync-enabled resources<br>• SkeletonActiveSyncResourceAdapter.java for Active Sync-only resources<br>• test.SkeletonResourceTest.java to create unit tests for a custom adapter |
| test | REF/test | Sample resource adapter test source files that you can use as a basis for a custom adapter. |
| thirdpartysource | REF/thirdpartysource | |
| transactionsigner | REF/transactionsigner | Sample implementation of a transactionsigner PKCS11KeyProvider. |
| BeforeYouBegin.README | REF | Outlines information you must collect before you customize an adapter. |
| build.xml | REF | Sample Ant Build script for building, testing, and distributing a project. |
| Design-for-Resource-Adapters.htm | REF | Document that describes the basic architecture and design of a resource adapter. |
| README | REF | Document describing the Sun Identity Manager REF Kit. |
| Waveset.properties | REF/config | Copy of the properties file you need to test a custom adapter. |

1. You can use the Resource Adapter Wizard to create an adapters for table-based resources instead of writing a custom adapter. See the "Configuration" chapter in *Identity Manager Administration* for more information about using this wizard.

# Set Up the Build Environment

This section contains instructions for setting up your build environment.

- On Windows
- On UNIX

➤ **Prerequisites:**

You must install the JDK version required for your Identity Manager version. See "Supported Software and Environments" in the *Identity Manager Release Notes* for information.

After installing the JDK, you must install the REF Kit by copying the entire `/REF` directory to your system.

## On Windows

If you are working on Microsoft Windows operating system, use the following steps to set up your build environment:

1. Change directories to a new directory.
2. Create a file called `ws.bat`.
3. Add the following lines to this file:

:

```
set WSHOME=<Path where REF Kit is installed>
set JAVA_HOME=<Path where JDK is installed>
set PATH=%PATH%;%JAVA_HOME%\bin
```

Where you set:

- `WSHOME` to the path to where the REF Kit is installed.
- `JAVA_HOME` to the path to where the JDK is installed.

4. Save and close the file.

### On UNIX

If you are working on a UNIX operating system, use the following steps to set up your build environment:

1. Change directories to a new directory.
2. Create a file called `ws.sh`.
3. Add the following lines to this file:

:

```
WSHOME=<path_where_REF_is_installed>
JAVA_HOME=<path_where_JDK_is_installed>
PATH=$JAVA_HOME/bin:$PATH

export WSHOME JAVA_HOME PATH
```

   Where you set:

   - `WSHOME` to the path to where the REF Kit is installed.
   - `JAVA_HOME` to the path to where JDK is installed.

4. Save and close the file.

# Writing Custom Adapters

After finishing the preparation work described in "Preparing for Adapter Development," you are ready to start writing your custom adapter.

This section describes how to write a custom adapter, including:

# Process Overview

The following sections provide a high-level overview of the steps you perform to create a custom adapter:

- How To Write a Standard Resource Adapter
- How To Write an Active Sync-Enabled Resource Adapter

## How To Write a Standard Resource Adapter

This section describes the processes to follow when creating a standard adapter or an Active Sync-enabled adapter.

**NOTE** The steps for writing a standard adapter are slightly different based on which operating system you are using.

Use the following steps to create a standard adapter:

1. Open a command window and go to the following directory:

   `\waveset\idm\adapter\src`

2. Rename the `SkeletonStandardResourceAdapter.java` skeleton file to a file name of your choice. See "Rename the Skeleton File" on page 137 for more information.

3. Edit the new adapter's source file as described in "Edit the Source File" on page 138.

4. Source the file you created previously to set up your environment:

   - **For Windows**: Source the `ws.bat` file.
   - **For Unix**: Source the `ws.sh` file.

5. Type the following command to compile your source files:

   - **For Windows**: `javac -d . -classpath %CLASSPATH% yourfile.java`
   - **For Unix**: `javac -d . -classpath $CLASSPATH yourfile.java`

### How To Write an Active Sync-Enabled Resource Adapter

This section describes the general steps you follow to create a custom Active Sync-Enabled adapter

If you are working on Microsoft Windows operating system, use the following steps to create a custom Active Sync-Enabled adapter:

1. Open a command window and change to the following directory:

   `\waveset\idm\adapter\src`

2. Rename (or copy) one of the following skeleton files to a file name of your choice. See "Rename the Skeleton File" on page 137 for more information.

   - `SkeletonStandardAndActiveSyncResourceAdapter.java` (for standard and Active Sync-enabled resources)
   - `SkeletonActiveSyncResourceAdapter.java` (for Active Sync-only resources)

3. Edit the new adapter's source file as described in "Edit the Source File" on page 138.

4. Source the file you created previously to set up your environment:

   - **For Windows**: Source the `ws.bat` file.
   - **For Unix**: Source the `ws.sh` file.

5. Type the following command to compile your source files:

   - **For Windows**: `javac -d . -classpath %CLASSPATH% yourfile.java`
   - **For Unix**: `javac -d . -classpath $CLASSPATH yourfile.java`

## Rename the Skeleton File

You must rename the skeleton adapter to a name that is appropriate for your new adapter. Perform the following actions:

- Rename the sample java file to match your new class name.
- Edit the source code to replace the sample class name with the new class name.

# Edit the Source File

After renaming the skeleton file, you must edit the new adapter's source code to replace specific text strings and to define default values you want the adapter to support.

Edit the adapter source file as follows:

1. Search for, and replace, the following text strings to determine where you must make adapter-specific modifications in the code.
   - `change-value-here` strings indicate where you must enter a substitution.
   - `@todo` strings indicates where you must rewrite a method for a particular scenario you are supporting.
2. Name the resource adapter type.

   This name displays in the New Resources menu in the Identity Manager Administrator interface.
3. Map the incoming resource attributes to Identity Manager account attributes by replacing default values in the `prototypeXML` string with your own default values for this adapter type. For example, you might want to delete the `RA_GROUPS` attribute from your adapter type.

   See "Map the Attributes " on page 139 for more information.
4. Add or delete methods from the skeleton file. In particular, you must add Java code to support `join`, `leave`, and `move` operations, which are not supported in this example file.

   See "Write the Adapter Methods" on page 141 for more information.
5. After editing the adapter file, you can load it into Identity Manager.

# Map the Attributes

Generally, you set options for the adapter type by mapping the incoming resource attributes to the standard Identity Manager account attributes or by creating your own custom attributes (called *extended schema* attributes).

Your resource must define resource attributes and set default values for resource attributes for which it makes sense to have default. The resource does not have to present the `prototypeXML` object.

**NOTE** The attributes in `SkeletonActiveSyncResourceAdapter` are mandatory. Do not delete these attribute definitions when customizing the file.

## Mapping Resource Attributes to Standard Account Attributes

To map incoming resource attributes to one of the standard Identity Manager account attributes, use the syntax shown in the following example.

**Code Example 2-6** Mapping a Resource Attribute

```
"<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"+
```

Where:

- The `<AccountAttributesTypes>` element surrounds the `prototypeXML` string where you map the resource attribute to the Identity Manager account attribute.
- The `<AttributeDefinitionRef>` element identifies the Identity Manager account attribute.

The following table describes the <AttributeDefinitionRef> element fields.

**Table 2-14** <AttributeDefinitionRef> Element Fields

| Element Field | Description |
|---|---|
| name | Identifies the Identity Manager account attribute to which the resource attribute is being mapped. (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 the skeleton file, replace change-value-here with the resource attribute name. |
| mapType | Identifies the incoming attribute type, which can be string, int, or encrypted. |

For more information on mapping resource attributes to account attributes, see "Map the Attributes " on page 139.

## Mapping Resource Attributes to Extended Schema Attributes

To map incoming resource attributes to attributes other than a standard Identity Manager attribute, you must create an *extended schema attribute*. The following example illustrates how to map a resource attribute to an extended schema attribute.

**Code Example 2-7** Mapping a Resource Attribute to an Extended Schema Attribute

```
<AccountAttributeType name='HomeDirectory' type='string'
   mapName='HomeDirectory' mapType='string'>\n"+
</AccountAttributeType>\n"+
```

You do not have to declare an ObjectRef type. The mapName field identifies the custom account attribute HomeDirectory. You define the mapType field the same as you would when mapping an attribute to a standard account attribute.

# Specify the Identity Template

You must use an identity template (or an account DN) to uniquely identify every user and group in your enterprise.

DNs display on the following Identity Manager User interface pages:

- Resources
- Distinguished Name Template
- Edit Schema

For more information about identity templates, see "Identity Template" on page 121.

# Write the Adapter Methods

The Identity Manager adapter interface provides general methods that you must customize to suit your particular environment. This section briefly describes:

- How to Write Standard Resource Adapter-Specific Methods
- How to Write Active Sync-Enabled Adapter Methods

## How to Write Standard Resource Adapter-Specific Methods

Standard resource adapter-specific methods are specific to the resource you are updating to synchronize with Identity Manager.

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

**NOTE** When writing a custom adapter

- Call the `setdisabled()` method on any `WSUser` object that is returned by a custom method.
- If the adapter implements the `AsynchronousResourceAdapter` class, then note that this adapter might be working with users that are partially initialized. (These users are created outside Identity Manager, but not fully populated with attributes.) The Provisioner does not automatically convert a Create operation to an Update operation if the WSUser already exists on the resource. Your resource adapter must distinguish this case.

### *Creating the Prototype Resource*

The following table describes the methods used to create a Resource instance.

**Table 2-15** Methods Used to Create a Resource Instance

| Method | Description |
|---|---|
| `staticCreatePrototypeResource` | Creates a Resource instance, usually from the predefined `prototypeXML` 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.<br>If extending an existing adapter, you can add resource attributes to specify different default values programmatically based on the super class's prototype resource. |

### *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 following table describes methods you can use to verify that your adapter is communicating with the resource and that the authorized account has access.

**Table 2-16** 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:<br>• Can basic connectivity to the resource be established?<br>• Does the account already exist?<br>• Do the account attribute values comply with all (if any) resource-specific restrictions or policies that have not been checked at a higher level?<br>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.<br>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.<br>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.<br>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:<br>• Can basic connectivity to the resource be established?<br>• Does the account already exist?<br>• Do the account attribute values comply with all (if any) resource-specific restrictions or policies that haven't been checked at a higher level?<br>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.<br>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 2-17** 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 2-18** 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 2-18** 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 2-19** 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 2-20** 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 2-21** Creating Accounts on the Resource

| Method | Description |
|---|---|
| `realCreate()` | Creates the account on the resource.<br>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 2-22** Deleting Accounts on the Resource

| Method | Description |
|---|---|
| `realDelete()` | Deletes an account(s) on the resource.<br>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 2-23** Updating Accounts on the Resource

| Method | Description |
|---|---|
| `realUpdate()` | Updates a subset of the account attributes.<br>By default, this method creates a connection, calls `realUpdate`, and closes the connection for each user object in the list.<br>**NOTE**: User account attributes from the resource are merged with any new changes from Identity Manager. |

**Table 2-24** Getting User Information

| Method | Description |
|---|---|
| `getUser()` | Retrieves information from the resource about user attributes.<br>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 2-25** List Methods

| Method | Description |
|---|---|
| `getAccountIterator()` | Used to discover or import all the users from a resource.<br>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.<br>Implement this method to generate lists that are used by the resource, such as a list of resource groups or distribution lists.<br>This method is called from the user form (not called by provisioning engine). |

➤ **Best Practice**

When writing an `AccountIterator` interface implementation for a custom adapter, try to do the following:

- Have the `AccountIterator.next()` method return a user that contains all attributes in the schema map when `getAccountIterator()` is called.
  The reconciler will trim the schema (in a cloned resource used for the `getAccountIterator()` request) to request only those attributes the reconciler needs. Generally, the reconciler needs only the `accountId` attribute; but there are cases when the reconciler has additional attributes in the schema. Other `getAccountIterator()` users, such as Load From Resource, potentially need all of the schema attributes.

  Try to create a "smart" adapter that does the right thing based on what is in the schema map. If your adapter can just list the accounts and get the requested information, then the adapter should just do those tasks. Otherwise, the adapter might have to fetch an account to get the required attributes. Adapters that are not smart always get all the attributes.
- Make your adapter as scalable as possible, which generally means the adapter does not list or fetch all of the accounts at once. Instead, your adapter should iterate over the accounts as the `AccountIterator.next()` method is called. Avoid having your adapter do much in the `AccountIterator` constructor.

The following example shows code for retrieving information from a resource and converting that information into information that Identity Manager can work with.

**Code Example 2-8** 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 2-26** Enable and Disable Methods

| Method | Description |
|---|---|
| `supportsAccountDisable()` | Returns `true` or `false` depending on whether the resource supports native account disable. |
| `realEnable()` | Implements native calls that are needed to enable the user account on the resource. |
| `realDisable()` | Implements native calls that are needed 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 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 Administrative 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 user name or password is invalid), then:

```
throw new WavesetException("Authentication failed for " + uid + ".");
```

## How to Write Active Sync-Enabled Adapter Methods

Active Sync-specific methods provide the mechanism for updating Identity Manager, which is the primary purpose of your Active Sync-enabled adapter adapter. These methods are based on pulling information from the authoritative resource. In addition, you use these methods to start, stop, and schedule the adapter.

The methods you are going to write in this section of the adapter are based on generic methods supplied with the skeleton adapter file. You must edit some of these methods, which are categorized by task.

The following sections describe general guidelines for creating Active Sync-enabled adapter methods:

- 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** You use the following scheduling parameters in Active Sync-enabled adapters:

- `RA_SCHEDULE_INTERVAL`
- `RA_SCHEDULE_INTERVAL_COUNT`

- RA_SCHEDULE_START_TIME
- RA_SCHEDULE_START_DATE

See Table 2-7 on page 117 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.

The following table describes the usage of scheduling parameters using some sample polling scenarios.

**Table 2-27** 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 following example shows how to retrieve the attribute and pass the update through to the base.

**Code Example 2-9** 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. Identity Manager calls the shutdown method, which is an opportunity for your adapter to cleanup any objects still in use from the polling loop.

## Configure the Adapter to Support Pass-Through Authentication

Identity Manager uses pass-through authentication to grant users and administrators access through one or more different passwords. Identity Manager manages pass-through authentication through the implementation of:

- Login applications (collection of login module groups)
- Login module groups (ordered set of login modules)
- Login modules (sets authentication for each assigned resource and specify one of several success requirements for authentication)

You configure a custom adapter to support pass-through authentication by

- Implementing the `authenticate()` method appropriately
- Including the `account.LOGIN` feature in the `getFeatures()` method map (`com.waveset.adapter.ResourceAdapter.ACCOUNT_LOGIN`)
- Including the `<LoginConfigEntry>` section in the resource's `prototypeXML`

When configuring a custom resource adapter to support an interactive login, you must enable the adapter to request additional information from a user during log in and after that user submits the initial login page.

The adapter `authenticate()` method controls whether the login becomes interactive. The `authenticate()` method's return values trigger the interactive login so the `authenticate()` is called again with the results of the next login page until the `authenticate()` method decides the login

- Fails by throwing an exception
- Succeeds by returning the account ID of the authenticated account in the `WavesetResult` as usual

To be interactive, the adapter must return a `WavesetResult` that

- *Does not* contain `ResultItems` with a `Constants.AUTHENTICATED_GUID` type or `Constants.AUTHENTICATED_IDENTITY` type
- *Does* contain `ResultItems` that are used to dynamically build a form for the next page of the login

Each `ResultItem` corresponds to a field in the form. `ResultItems` must have the `Constants.CONTINUE_AUTHENTICATION_ATTR` type with values in the following format:

*`label`*`|`*`attrName`*`|`*`displayType`*`|`*`prompt`*`|'`*`isId`*`'`

Where

- *label* is a string containing a label or `none`.
- *attrName* is the login attribute name that is passed into the next `authenticate()` method call as a key in the `loginInfo HashMap`.
- *displayType* describes the type of form field to use. *displayType* values include
  - `text`
  - `secret`
  - `label`
  - `checkbox`

- *prompt* corresponds to the title or label of the form field.
- *isId* is an optional string.

  If you use the `isId` string, the value of the form field is added to the `loginInfo HashMap` with the key `Constants.ACCOUNT_ID` and the value of the field.

The following `ResultItem` types are also "round-tripped" and returned in the `loginInfo HashMap` on the next `authenticate()` call:

- `Constants.CONTINUE_AUTHENTICATION_ACCOUNT_HANDLE` keep track of which user or account is in the process of being authenticated.
- `Constants.CONTINUE_AUTHENTICATION_PREVIOUS_ATTR` remove previous authentication attributes from the `loginInfo`, so the `loginInfo` does not contain an "old" authentication `attr`.

# Define the Resource Object Components

This section describes how to define the following resource object components:

- Defining Resource Object Classes
- Defining Resource ObjectTypes
- Defining Resource Object Features
- Defining Resource Object Attributes

## Defining Resource 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 the following example, 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 2-10** 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 the next example, 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 2-11** Using `groupOfUniqueNames` Object Class

```
<ObjectClasses primary='groupOfUniqueNames' operator='OR'>\n"+
   <ObjectClass name='groupOfNames'/>\n"+
   <ObjectClass name='groupOfUniqueNames'/>\n"+
</ObjectClasses>\n"+
```

In this example, only one object class is defined so all `create`, `get`, `list`, and `update` operations are performed using object class `organizationalUnit`.

**Code Example 2-12** Using `organizationalUnit` Object Class

```
<ObjectClasses operator='AND'>\n"+
   <ObjectClass name='organizationalUnit'/>\n"+
</ObjectClasses>\n"+
```

Because there is only one object class, you can exclude the `<ObjectClasses>` section. If you exclude the `<ObjectClasses>` section, the object class defaults to the `<ObjectType>` `name` attribute value. 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.

## Defining Resource ObjectTypes

Resource *Object types* uniquely define a specific type of resource, and you define object types in the adapter's `prototypeXML` string.

The XML `<ObjectTypes>` element is a container within the adapter's `prototypeXML` string that contains one or more object type definitions to be managed on that resource. This `<ObjectTypes>` element fully describes the resource-specific object to Identity Manager, including the following:

- A list of specific object classes contained in the object type (required only for LDAP-compliant directories)
- A list of supported features
- A list of object type-specific attributes that are available within Identity Manager for editing and searching.

The following table describes the supported attributes of the <ObjectType> element.

**Table 2-28** Supported <ObjectType> Element Attributes

| Attribute | Description |
|---|---|
| name | Defines the name by which this object type is displayed and referred to within Identity Manager (required). |
| icon | Defines the name of the .gif file to display in the Identity Manager interface for objects of this type. You must install this .gif file in idm/applet/images for use by Identity Manager. |
| container | Defines whether this type of resource object can contain other resource objects of the same type or of a different type.<br>• If true, this resource object type can contain other resource objects.<br>• If false, this resource object type cannot contain other resource objects. |

The following example shows ObjectType definitions:

**Code Example 2-13** Example ObjectType 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"+
```

## Defining Resource 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 can specify a form attribute, which defines the resource form used to process create and update features. If you do not specify a form attribute, Identity Manager processes the create and update features with the same form used by all resources of a given type.

The following table describes the object feature mappings.

**Table 2-29** 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 |

In the following example, the <ObjectFeatures> section includes all supported object features. Your resource adapter can support all of these features or just a subset of features. The more object features your adapter supports, the richer the object management function within Identity Manager.

**Code Example 2-14** <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"+
```

## Defining Resource Object Attributes

The <ObjectAttributes> section specifies the set of attributes to be managed and queried in Identity Manager. Each <ObjectAttribute> element name should be the same as the native resource attribute name. Unlike user attributes in Identity Manager, no attribute mapping is specified. Use only the native attribute names.

The following table describes attributes that are required for <ObjectAttributes>.

**Table 2-30** 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. |

The following example shows an <ObjectAttributes> section defined in an <ObjectType>.

**Code Example 2-15** <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"+
```

The following table describes the <ObjectAttribute> attributes.

**Table 2-31** <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 are responsible for coercing all string values into the appropriate type based on the resource attribute name.

## Defining Resource Forms

You must provide the following resource forms:

- A ResourceForm 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*

- 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*

You can also assign an *optional* form that processes incoming data before storing it in Identity Manager. This resource form is a mechanism that transforms incoming data from the schema map and applies the transformed data to the User view. The sample form also performs actions, such as enabling and disabling an account, that are based on specific incoming data values such as employee status.

The following table describes attributes contained in the top-level namespace.

**NOTE** All values are strings unless otherwise specified.

**Table 2-32** 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. |

# Installing Custom Adapters

To install a customized resource adapter:

1. If necessary, create the following directory:

   `idm/WEB-INF/classes/`*`package_path`*

   Where *package_path* is the package where your class is defined. For example,

   `com/waveset/adapter/sample`

2. Copy your *NewResourceAdapter*`.class` file into the directory you just created.

3. Create a `gif` image that is 18x18 pixels and 72 DPI in size to represent your adapter. Identity Manager displays this `.gif` file image next to the resource name on the List Resources page.

   You must use the following format when naming your `.gif` file:

   *`YourAdapterName`*`.gif`

   You must replace any spaces in your adapter name with underscores. For example, look at some of the existing adapter names in

   `\waveset\idm\web\applet\images`

4. Copy the `.gif` file to `idm/applet/images`.

5. Stop and restart the application server.
   For information about working with application servers, see *Identity Manager Installation*.

6. Create an HTML help file for your resource.

   **NOTE** The `idm.jar` in the `com/waveset/msgcat/help/resources` directory contains example help files.

   See *Identity Manager Workflows, Forms, and Views* for information about including online help with an application.

7. From the Managed Resources page of the Administrator interface, click the Custom Adapter button and enter the full class name of your adapter class. For example

   `com.waveset.adapter.sample.`*`NewResourceAdapter`*

8. Create a resource in Identity Manager using your adapter.

9. Ensure your native managed system is operational.
10. Test the connectivity of your new Identity Manager resource, as described in "Checking Connections and Operations" on page 143.

# Testing Custom Adapters

After writing a custom resource adapter, you must test the validity of that adapter. In particular, you must test the connection to the resource.

Topics covered in this section include:

- Unit Testing Your Adapter
- Compatibility Testing Your Adapter

## Unit Testing Your Adapter

Use the following steps to unit-test the validity of a custom adapter (in particular, to test the connection to the resource):

1. Save the adapter.
2. Run unit tests on that adapter from your own machine.
3. Load the adapter into Identity Manager.
4. Test the adapter in Identity Manager, as follows:
   a. Log into the Identity Manager Administrator interface.
   b. Click the Resources > List Resources tabs.
   c. Click Start on the List Resources page.

      The Start button is enabled only if the resource start-up type is Automatic or Manual.

# Compatibility Testing Your Adapter

Writing and maintaining a custom resource adapter can be a very complex process. Developers commonly discover that their custom adapters do not perform as expected, or that the adapters do not perform the functions expected by Identity Manager. Even well-written resource adapters will sometimes not work well after the external resource has been upgraded.

Identity Manager provides a compatibility testing mechanism that you can use to verify the quality of a custom resource adapter. This tester

- Makes it easier for you to write, publish, and maintain a custom adapter
- Makes it easier for you to run the adapter and interpret results
- Focuses on testing the adapter's supported features as fully as possible in an adapter-independent manner
- Simplifies troubleshooting an adapter

This section describes how to use Identity Manager's Compatibility Test Suite. The information is organized as follows:



## How the Compatibility Test Suite Works

Identity Manager's Compatibility Test Suite performs a set of standard tests to check the adapter's supported features. If a particular test requires a feature not provided by the adapter, Identity Manager skips that test.

The Compatibility Test Suite requires certain information, such as a valid user name and password to run a compatibility test on a resource adapter. You can typically use the standard *DataProvider* (provided with Identity Manager) to supply the data required for the test.

**NOTE** For special circumstances, such as when you want to provide information in a class instead of as an expression in XML, you can write a custom DataProvider.

## How to Run the Compatibility Tests

To run the Identity Manager Compatibility Test Suite, use the following steps:

1. Open a command window.
2. At the command prompt, type the `lh` command using the following format:

```
$WSHOME/bin/lh com.sun.idm.testing.adapter.CompatibilitySuite [Options] [testName]
```

Where:

- [*options*] include:
  - `-h`: Use to access usage information

    For example:

    ```
    Usage: CompatibilitySuite [arguments]

    Valid arguments:
    -propsFile value      Path to properties files
    -formatter value      Formatter to use for formatting output of tests
    -user value           Name of user to execute test as
    -pass value           Plain Text password used to log user on
    -import value         Comma separated list of files (on server) to import
    -toDir value          Directory to put test output in
    -v                    Echos all arguments passed in to screen
    -h                    Displays usage message
    ```

  - `-propsFile` *file*: Use to specify a properties file name.
  - `-formatter` *type,path*: Use to specify XML, HTML, or plain text and a path in which to put this file.
- [*testName*] is a comma-separated list of the tests to run.

The following properties control how tests are executed:

| Property | Description |
|---|---|
| `adapter` | Classname of the adapter to test |
| `dp` | Name of a custom DataProvider |
| `importScript` | Comma-separated list of paths to the scripts to execute<br>**Note**: These scripts return a string of imported XML. |
| `ns` | DataProvider namespace |
| `includedTests` | Comma-separated list of tests to include |
| `excludedTests` | Comma-separated list of tests to exclude |
| `import` | Comma-separated list of files to import |

You can specify these properties directly from the command line, or add them to a properties file specified from the command line. For example,

```
lh -DpropName=propValue
```

Where properties conflict, properties in the property file specified by `propsFile` are used.

**NOTE** When you use the `[testName]` command, the Compatibility Test Suite ignores the `includedTests` and `excludedTests` options.

In most cases, the framework provided by Identity Manager is flexible enough to test the resource adapter. However, you can easily extend the functionality in two places if necessary:

- You can implement the DataProvider interface to create a custom DataProvider. A custom DataProvider allows data to come from any source.
- You can implement the `CompatibilityHelper` interface to provide a `CompatibilityHelper`. A `CompatibilityHelper` provides a way to initialize a resource before running tests.

See the Javadoc for more information about implementing these interfaces and the required naming conventions.

## Example 1: Using the Default DataProvider to Run Compatibility Tests

This example illustrates how to run compatibility tests on a `SimulatedResourceAdapter` using the default DataProvider.

### *To Prepare the Test*

To prepare this compatibility test,

1. Set up the following files:

   `sample/compat/example.1/example.properties`

   `sample/compat/example.1/SimulatedCompatibilityConfig.xml`

   **NOTE** The default path to the simulated resource in `SimulatedCompatibilityConfig` is `/tmp/mySimulatedResource.xml`.

   You can edit this path if you want to specify a different location.

2. Before executing the example, copy `ant-junit.jar` from Apache ant 1.6.5 to your `$WSHOME/WEB-INF/lib` directory.

### *To Execute the Test*

Execute the compatibility test as follows:

1. Open a command window.
2. At the prompt, type

   ```
   cd $WSHOME
   ```

   ```
   bin/lh com.sun.idm.testing.adapter.CompatibilitySuite -propsFile
   sample/compat/example.1/example.properties
   ```

Your output should look similar to the following example:

**Code Example 2-16** Compatibility Test Results Using the Default DataProvider

```
TestSuite: com.sun.idm.testing.adapter.CompatibilitySuite
Starting internal database server ...
DB Server @ jdbc:hsqldb:hsql://127.0.0.1:57022/idm
Importing file sample/compat/example.1/SimulatedCompatibilityConfig.xml
'Create(com.sun.idm.testing.adapter.compatibility.Create)' skipped  (unknown)
'Authenticate(com.sun.idm.testing.adapter.compatibility.AuthenticateUser)' skipped
(unknown)
'DeleteExisting(com.sun.idm.testing.adapter.compatibility.DeleteExisting)' skipped
(unknown)
'UpdateExisting(com.sun.idm.testing.adapter.compatibility.UpdateExisting)' skipped
(unknown)
'RenameExisting(com.sun.idm.testing.adapter.compatibility.RenameExisting)' skipped
(unknown)
'EnableExisting(com.sun.idm.testing.adapter.compatibility.EnableExisting)' skipped
(unknown)
'DisableExisting(com.sun.idm.testing.adapter.compatibility.DisableExisting)' skipped
(unknown)
'Iterate(com.sun.idm.testing.adapter.compatibility.Iterate)' skipped  (unknown)
'DeleteMissing(com.sun.idm.testing.adapter.compatibility.DeleteMissing)' passed  (77 ms)

Tests run: 9, failures: 0, errors: 0, skipped: 8, Time elapsed: 10864 ms
```

### *What Happened*

In Code Example 2-16, the `lh` command runs the compatibility test with the following argument:

```
-propsFile sample/compat/example.1/example.properties
```

Both the `adapter` and `ns` properties are required to run the test.

- The `adapter` property provides the adapter class name to be tested.
- The `ns` property provides a namespace for the test.

  The DataProvider can use the namespace to set up multiple configurations.

Code Example 2-16 also uses the `import` property, which imports a list of files into the repository. The `import` property is similar to `lh import` *filename*.

When you start the compatibility test, the tester retrieves the `adapter` and `ns` properties from the specified properties.

The default DataProvider retrieves data from the extension element of a *namespace*`#TestData` configuration object, which in this example was `SimulatedCompatibilityConfig#TestData`.

**NOTE** If you do not specify a DataProvider when setting up a test, Identity Manager used the default DataProvider.

The DataProvider retrieves this `SimulatedCompatibilityConfig#TestData` configuration object from the repository.

To get the configuration object into the repository, you must define the object in the following file, which is specified in the `import` property:

```
sample/compat/example.1/SimulatedCompatibilityConfig.xml
```

To simplify configuration in Code Example 2-16, only one test was run with the `includedTests=DeleteMissing` parameter.

**NOTE** See the Javadoc for more information about which parameters are available and which parameters are required for the different tests.

## Example 2: Adding More Data

To run the creation tests, and other tests that create users, you must add more data to the configuration object. In this next example, you must use the default DataProvider again and import an XML file.

### *To Prepare the Test*

To prepare this compatibility test,

1. Set up the following files:

   ```
   sample/compat/example.2/example.properties
   ```

   ```
   sample/compat/example.2/SimulatedCompatibilityConfig.xml
   ```

   **NOTE** The default path to the simulated resource in `SimulatedCompatibilityConfig` is `/tmp/mySimulatedResource.xml`.

   You can edit this path if you want to specify a different location.

2. Before executing the example, copy `ant-junit.jar` from Apache ant 1.6.5 to your `$WSHOME/WEB-INF/lib` directory.

### *To Execute the Test*

Execute the compatibility test as follows:

1. Open a command window.
2. At the prompt, type

   ```
   cd $WSHOME
   ```

   ```
   bin/lh com.sun.idm.testing.adapter.CompatibilitySuite -propsFile
   sample/compat/example.2/example.properties
   ```

Your output should look similar to the following example:

**Code Example 2-17** Compatibility Test Results After Adding Tests

```
TestSuite: com.sun.idm.testing.adapter.CompatibilitySuite
Starting internal database server ...
DB Server @ jdbc:hsqldb:hsql://127.0.0.1:57022/idm
Importing file ./sample/compat/example.2/SimulatedCompatibilityConfig.xml
'Authenticate(com.sun.idm.testing.adapter.compatibility.AuthenticateUser)' skipped
(unknown)
'UpdateExisting(com.sun.idm.testing.adapter.compatibility.UpdateExisting)' skipped
(unknown)
'RenameExisting(com.sun.idm.testing.adapter.compatibility.RenameExisting)' skipped
(unknown)
'Iterate(com.sun.idm.testing.adapter.compatibility.Iterate)' skipped  (unknown)
'DeleteMissing(com.sun.idm.testing.adapter.compatibility.DeleteMissing)' passed  (15 ms)
'EnableExisting(com.sun.idm.testing.adapter.compatibility.EnableExisting)' passed  (259 ms)
'DisableExisting(com.sun.idm.testing.adapter.compatibility.DisableExisting)' passed  (7 ms)
'DeleteExisting(com.sun.idm.testing.adapter.compatibility.DeleteExisting)' passed  (3 ms)
'Create(com.sun.idm.testing.adapter.compatibility.Create)' passed  (3 ms)

Tests run: 9, failures: 0, errors: 0, skipped: 4, Time elapsed: 10178 ms
```

### *What Happened*

You requested additional tests by setting the following property in the properties file:

```
IncludedTests=DeleteMissing,Create,EnableExisting,DisableExisting,DeleteExisting
```

These tests required more data from the DataProvider.

To provide this new data, several changes were made to the configuration object specified by `SimulatedCompatibilityConfig.xml`.

The `SimulatedCompatibilityConfig.xml` file added a `create` attribute containing a username, password, and list of user attributes. The default DataProvider uses the `create` attribute when the compatibility tests ask for the username, password, and attributes required to create a single user.

The `SimulatedCompatibilityConfig.xml` file also added a schema map.

## Example 3: Finishing the Test Configuration

In this next example, you finish the test configuration.

### *To Prepare the Test*

To finish the compatibility test configuration,

1. Set up the following files:

   `sample/compat/example.3/example.properties`

   `sample/compat/example.3/SimulatedCompatibilityConfig.xml`

   **NOTE** The default path to the simulated resource in `SimulatedCompatibilityConfig` is `/tmp/mySimulatedResource.xml`.

   You can edit this path to specify a different location by changing two lines in the file.

2. Before executing the example, copy `ant-junit.jar` from Apache ant 1.6.5 to your `$WSHOME/WEB-INF/lib` directory.
3. You must initialize the repository to run the `encrypt` command.

For example, use the `lh import sample/init.xml` command to initialize the repository, where the original file looks like the following:

```
<Attribute name="login_infos">
         <List>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password" value="ctPass" />
             <Attribute name="shouldfail" value="no" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password" value="wrongPass" />
             <Attribute name="shouldfail" value="yes" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password">
               <!-- result of 'encrypt ctPass' from lh console -->

<EncryptedData>11D1DEF534EA1BE0:-32DFBF32:1165DC91D73:-7FFA|mDBIkSQB3xg=</EncryptedData>
             </Attribute>
             <Attribute name="shouldfail" value="no" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password">
               <!-- result of 'encrypt wrongPass' from lh console -->

<EncryptedData>11D1DEF534EA1BE0:-32DFBF32:1165DC91D73:-7FFA|m0n9bAaMx+sKpqs5PmH3eQ==
</EncryptedData>
             </Attribute>
             <Attribute name="shouldfail" value="yes" />
           </Object>
         </List>
       </Attribute>
```

4. In each case, use an `encrypt` command from the `lh console` to get an encrypted password that can be decrypted in your environment.

   Run `lh console` and at the console prompt, type the text in single quotes for each of the preceding `EncryptedData` entries (for example, `encrypt ctPass`) and replace the text between `<EncryptedData>` and `</EncryptedData>` with the result.

See the following example:

```
<!-- result of 'encrypt ctPass' from lh console -->
<EncryptedData>11D1DEF534EA1BE0:-65F64461:1163AB5A7B2:-7FFA|iMm4Tcqck+M=</EncryptedData>

<!-- result of 'encrypt wrongPass' from lh console -->
<EncryptedData>11D1DEF534EA1BE0:-65F64461:1163AB5A7B2:-7FFA|d1/PheqRok+J3uaggtj9Gw==
</EncryptedData>
```

Alternatively, you can have the DataProvider skip the two `login info` entries by commenting out the whole block as follows:

```
<!-- commented out
       <Attribute name="login_infos">
         <List>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password" value="ctPass" />
             <Attribute name="shouldfail" value="no" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password" value="wrongPass" />
             <Attribute name="shouldfail" value="yes" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password">

<EncryptedData>11D1DEF534EA1BE0:-32DFBF32:1165DC91D73:-7FFA|mDBIkSQB3xg=</EncryptedData>
             </Attribute>
             <Attribute name="shouldfail" value="no" />
           </Object>
           <Object>
             <Attribute name="sim_user" value="ctUser" />
             <Attribute name="sim_password">

<EncryptedData>11D1DEF534EA1BE0:-32DFBF32:1165DC91D73:-7FFA|m0n9bAaMx+sKpqs5PmH3eQ==
</EncryptedData>
             </Attribute>
             <Attribute name="shouldfail" value="yes" />
           </Object>
         </List>
       </Attribute>
-->
```

5. Next, copy the new data and paste it inside the `<EncryptedData>` tag to replace the old data. Be certain there are no extra spaces or line breaks inside the tag.

### *To Execute the Tests*

Run the tests again as follows:

1. Open a command window.
2. At the prompt, type

   ```
   cd $WSHOME
   ```

   ```
   bin/lh com.sun.idm.testing.adapter.CompatibilitySuite -propsFile
   sample/compat/example.3/example.properties
   ```

Your output should look similar to the following example:

**Code Example 2-18** Compatibility Test Results After Finishing the Test Configuration

```
TestSuite: com.sun.idm.testing.adapter.CompatibilitySuite
Starting internal database server ...
DB Server @ jdbc:hsqldb:hsql://127.0.0.1:57022/idm
Importing file ./sample/compat/example.3/SimulatedCompatibilityConfig.xml
'Create(com.sun.idm.testing.adapter.compatibility.Create)' passed  (31 ms)
'Authenticate(com.sun.idm.testing.adapter.compatibility.AuthenticateUser)' passed  (12 ms)
'DeleteExisting(com.sun.idm.testing.adapter.compatibility.DeleteExisting)' passed  (1 ms)
'DeleteMissing(com.sun.idm.testing.adapter.compatibility.DeleteMissing)' passed  (1 ms)
'UpdateExisting(com.sun.idm.testing.adapter.compatibility.UpdateExisting)' passed  (33 ms)
'RenameExisting(com.sun.idm.testing.adapter.compatibility.RenameExisting)' passed  (5 ms)
'EnableExisting(com.sun.idm.testing.adapter.compatibility.EnableExisting)' passed  (10 ms)
'DisableExisting(com.sun.idm.testing.adapter.compatibility.DisableExisting)' passed  (5 ms)
'Iterate(com.sun.idm.testing.adapter.compatibility.Iterate)' passed  (352 ms)

Tests run: 9, failures: 0, errors: 0, skipped: 0, Time elapsed: 10262 ms
```

### *What Happened*

The line specifying the included tests was removed from the `example.properties` file, which should run the entire suite.

Additional data is required for the remaining tests, so the `SimulatedCompatibilityConfig.xml` file was modified to include `update`, `rename`, and `iterate` attributes. These attributes modify users, rename users, and create a set of users over which to iterate. In addition, the file added an `login_info` attribute that specifies a list of items used to authenticate a user if authentication is supported by the resource adapter.

Finally, the `shouldfail` attribute, provided in the file under the `login info` entries, allows negative tests. The tests in the suite should now complete with no errors or skipped tests.

## Example 4: Executing Javascript or Beanshell Script

The compatibility tests enable you to execute a javascript or beanshell script and then import the script results into the repository. Either script must return a string that contains the XML to be imported.

Identity Manager provides an example Apache Velocity template and some supporting beanshell script that uses the template. The beanshell script was created just to fill in the required variables, which makes it very easy to work with the default DataProvider.

### *To Prepare the Test*

To prepare the compatibility test to execute with a beanshell script

1. Set up the following files:

   `sample/compat/example.4/example.properties`

   `sample/compat/example.4/SimulatedCompatibilityConfig.bsh`

   **NOTE** The default path to the simulated resource in `SimulatedCompatibilityConfig` is `/tmp/mySimulatedResource.xml`.

   You can edit this path if you want to specify a different location.

   You must change two lines in the file.

2. Before executing the example, copy `ant-junit.jar` from Apache ant 1.6.5 to your `$WSHOME/WEB-INF/lib` directory.

### *To Execute the Tests*

Execute the compatibility tests as follows:

1. Open a command window.
2. At the prompt, type

   ```
   cd $WSHOME
   ```

   ```
   bin/lh com.sun.idm.testing.adapter.CompatibilitySuite -propsFile
   sample/compat/example.4/example.properties
   ```

Your output should look similar to the following example:

**Code Example 2-19** Compatibility Test Results After Executing Beanshell Script

```
TestSuite: com.sun.idm.testing.adapter.CompatibilitySuite
Starting internal database server ...
DB Server @ jdbc:hsqldb:hsql://127.0.0.1:57022/idm
Executing script
/opt/build/dv207518/adapterTestsTemp/waveset/export/pipeline/./sample/compat/example.4/Simu
latedCompatibilityConfig.bsh
Importing results
'Create(com.sun.idm.testing.adapter.compatibility.Create)' passed  (25 ms)
'Authenticate(com.sun.idm.testing.adapter.compatibility.AuthenticateUser)' passed  (11 ms)
'DeleteExisting(com.sun.idm.testing.adapter.compatibility.DeleteExisting)' passed  (5 ms)
'DeleteMissing(com.sun.idm.testing.adapter.compatibility.DeleteMissing)' passed  (4 ms)
'UpdateExisting(com.sun.idm.testing.adapter.compatibility.UpdateExisting)' passed  (4 ms)
'RenameExisting(com.sun.idm.testing.adapter.compatibility.RenameExisting)' passed  (3 ms)
'EnableExisting(com.sun.idm.testing.adapter.compatibility.EnableExisting)' passed  (11 ms)
'DisableExisting(com.sun.idm.testing.adapter.compatibility.DisableExisting)' passed  (5 ms)
'Iterate(com.sun.idm.testing.adapter.compatibility.Iterate)' passed  (22 ms)

Tests run: 9, failures: 0, errors: 0, skipped: 0, Time elapsed: 11354 ms
```

### *What Happened*

The DataProvider supplied an `importScript` property, which caused the `SimulatedCompatibilityConfig.bsh` script to run. This script returns an XML string that is imported into the repository as a configuration object. The script specified the necessary items, and the velocity template creates the string.

You can use one of the following methods to debug the `import` script:

- Use `lh console` to turn on tracing and then check the generated log files for the script's return value. For example, type:

  ```
  trace 4 com.sun.idm.testing.adapter.CompatibilitySuite
  ```

- Use the `excludedTests` property and exclude each test. No tests run, but the script executes.

**NOTE** This example used beanshell scripting, but you can also use Javascript.

Several beanshell helpers are provided in the `sample/compat/beanshell` directory to make scripting easier by using the Apache Velocity template engine.

**NOTE** Commented examples are included to help you use the beanshell helpers.

To use the templates, add the following code at the top of your beanshell script:

```
// import helpers
String wavesetHome = Util.getWavesetHome();

if(wavesetHome != null) {
    if ( wavesetHome.startsWith("file:" ) ) {
        wavesetHome = wavesetHome.substring("file:".length());
    }

    addClassPath(wavesetHome + "./sample/compat/");
}

importCommands("beanshell");
```

Using the helpers is optional.

When using a script, the only requirement is that the script must return a string containing XML. The script can access any of the parameters that were passed into the `CompatibilitySuite` using `_params`.

Where `_params` can contain any of the following properties.

| Property | Description |
|---|---|
| `adapter` | Classname of the adapter to test |
| `dp` | Name of a custom DataProvider |
| `importScript` | Comma-separated list of paths to the scripts to execute |
| | **Note**: These scripts return a string of imported XML. |
| `ns` | DataProvider namespace |
| `includedTests` | Comma-separated list of tests to include |
| `excludedTests` | Comma-separated list of tests to exclude |
| `import` | Comma-separated list of files to import |

**NOTE** These properties are not provided in the _params map unless you set them in the properties file or used the -D command from the command line to add these properties to the _params map.

In beanshell, you can use a call to params.get("parameter_name") to retrieve these parameters.

If the beanshell script needs to know how the namespace parameter was set so that the script could form the name of the configuration object, the parameter would be retrieved as follows:

```
String namespace = _params.get("ns");
```

## Example 5: Running Tests from Inside the Web Container

Use the following process to run compatibility tests from inside the web container.

### *To Prepare the Test*

Before executing the example,

1. Copy ant-junit.jar from Apache ant 1.6.5 to your $WSHOME/WEB-INF/lib directory.
2. Enable the com.sun.idm.testing.adapter.compatibility.CTServlet by uncommenting the following in the web.xml file:
   - Uncomment servlet definition:

```
<servlet>
    <servlet-name>CompatibilityTests</servlet-name>

<servlet-class>com.sun.idm.testing.adapter.compatibility.CTServlet<
/servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

- Uncomment servlet mapping:

```
<servlet-mapping>
    <servlet-name>CompatibilityTests</servlet-name>
    <url-pattern>/servlet/CTServlet</url-pattern>
  </servlet-mapping>
```

**NOTE**

- You might have to restart your computer to use the new servlet.
- The servlet accepts `POST` requests with certain parameters. Some parameters, such as imported files, allow you to specify multiples.

**3.** You can specify the following parameters to the Compatibility Suite:

| Property | Description |
|---|---|
| `adapter` | Classname of the adapter to test |
| `dp` | Name of a custom DataProvider |
| `excludedTests` | Comma-separated list of tests to exclude |
| `import` | Comma-separated list of files to import |
| `importScript` | Comma-separated list of paths to the scripts to execute<br>**Note**: These scripts return a string of imported XML. |
| `includedTests` | Comma-separated list of tests to include |
| `ns` | DataProvider namespace |
| `pass` | Plain Text password used to log user on<br>Note: This password is sent in plain text, which may influence your decision on whether or not to enable the servlet. |
| `user` | Name of user who executes test |

Additional, remote-only parameters include:

| Property | Description |
|---|---|
| `importXMLText` | String containing XML to import |
| `importScriptText` | String containing script to run |
| `importScriptSuffix` | Specify `bsh` if the script is a beanshell script<br>Specify `js` if the script is javascript.<br>**Note**: If you specify multiple scripts to the servlet, the scripts must all be javascript or all beanshell, you cannot specify one of each. |

You can access the servlet through `debug/CompatTests.jsp` or the command line Java program, `CTContainerTest.java`.

4. To prepare for running the tests remotely, copy the file `idmtesting.jar` and the example folders under `sample/compat` to the remote system.

### *To Execute the Tests*

To run the tests from the `CompatTests.jsp` page,

1. Open a browser and navigate to your *idm instance*`/debug/CompatTests.jsp`. For example,

   `http://`*`example.com`*`:8080/idm/debug/CompatTests.jsp`

2. To run the example, you must provide the following values:

   `Namespace = SimulatedAdapterTests`

   `Adapter = com.waveset.adapter.SimulatedResourceAdapter`

   `User Name to Run Test as = configurator`

   `Password for User to Run Test as =` *configurator's password*

   `Script type = Beanshell radio button`

   `Import Result of this Script Text =` *SimulatedCompatibilityConfig.bsh file contents*

3. Copy the contents of the `SimulatedCompatibilityConfig.bsh` file from the `sample/compat/example.4` directory and paste them into this text field.

**NOTE** This script runs Example 4, but you can run the other examples in the same way. The other parameters are available too, but the names are slightly altered in the `jsp` file.

You can also run the compatibility test remotely from a command line Java program called `CTContainerTest`. The usage is as follows:

`CTContainerTest -url url [-v] [-parm1_name parm1_value -parm2_name parm2_value ... -parmx_name parmx_value]`

Where:

- Parameter names are the same as parameters accepted by the servlet.
- Parameter values are the same as values accepted by the servlet.

The servlet does not support the following parameters as a command line argument:

- `importScriptText`

  You can use the `importLocalScriptFile` parameter to send the `importScriptText` parameter to the servlet. The `importLocalScriptFile` option reads the contents of the file specified by the parameter value, and submits the content of that file to the servlet with the `importScriptText` parameter name.

- `importXMLText`

  You can use the `importLocalXMLFile` parameter to send the `importXMLText` parameter to the servlet. The `importLocalXMLFile` option reads the contents of the file specified by the parameter value, and submits the content of that file to the servlet with the `importXMLText` parameter name.

For more information about the available parameters and their use, run the `CTContainerTest` program with no arguments, as follows:

```
java -cp idmtesting.jar com.sun.idm.testing.adapter.CTContainerTest
```

The following examples illustrate different ways to run this command.

**NOTE** To run these examples in a Windows environment, you must adjust the *hostname* and *port*, change the classpath separation character from a colon (:) to a semicolon (;), and change the path separator from a backward slash (/) to a forward slash (\).

**Code Example 2-20** Running a Compatibility Test Using the Default DataProvider

```
java -cp idmtesting.jar:idmcommon.jar
com.sun.idm.testing.adapter.CTContainerTest -url
"http://host:port/idm/servlet/CTServlet" -adapter
com.waveset.adapter.SimulatedResourceAdapter -ns SimulatedAdapterTests
-importLocalXMLFile ./example.1/SimulatedCompatibilityConfig.xml
-includedTests DeleteMissing
```

**Code Example 2-21** Running a Compatibility Test with Added Tests

```
java -cp idmtesting.jar:idmcommon.jar
com.sun.idm.testing.adapter.CTContainerTest -url
"http://host:port/idm/servlet/CTServlet" -adapter
com.waveset.adapter.SimulatedResourceAdapter -ns SimulatedAdapterTests
-importLocalXMLFile ./example.2/SimulatedCompatibilityConfig.xml
-includedTests
DeleteMissing,EnableExisting,DisableExisting,DeleteExisting,Create
```

**Code Example 2-22** Running a Compatibility Test After Finishing the Test Configuration

```
java -cp idmtesting.jar:idmcommon.jar
com.sun.idm.testing.adapter.CTContainerTest -url
"http://host:port/idm/servlet/CTServlet" -adapter
com.waveset.adapter.SimulatedResourceAdapter -ns SimulatedAdapterTests
-importLocalXMLFile ./example.3/SimulatedCompatibilityConfig.xml
```

**Code Example 2-23** Running a Compatibility Test After Executing Beanshell Script

```
java -cp idmtesting.jar:idmcommon.jar
com.sun.idm.testing.adapter.CTContainerTest -url
"http://host:port/idm/servlet/CTServlet" -adapter
com.waveset.adapter.SimulatedResourceAdapter -ns SimulatedAdapterTests
-importLocalScriptFile ./example.4/SimulatedCompatibilityConfig.bsh
```

# Testing the Resource Object

This section describes the following methods for testing the resource object:

- Viewing and Editing a Resource Object
- Testing the Resource Object in Identity Manager

## Viewing and Editing a Resource Object

You can confirm the configuration of your resource by viewing the raw XML in the repository.

Use the following steps to view and edit a resource object:

1. Log into the Administrator user interface.
2. Open the Identity Manager Debug pages by entering `http://host:port/idm/debug` in the browser.
3. Choose Resource from the pull-down menu located next to the List Objects button.
4. Click the List Objects button.

   The List Objects of Type: Resource page displays with a list of all resource adapters and Active Sync-enabled adapters.

**NOTE** All resource adapter and Active Sync-enabled adapter classes are based on existing Identity Manager Resource classes.

5. Find the resource object you want to see.
   - To view the resource object, click the View link.
   - To edit the resource object, click the Edit link.
6. When you are finished, click Back.

## Testing the Resource Object in Identity Manager

You can use the Find Resources and List Resources pages in the Identity Manager Administrative interface to test your implementation of a resource object.

- Select Resource > List Resource to confirm the following performance characteristics:

**Table 2-33** 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<br>• 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 2-34** 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. |

# Troubleshooting Custom Adapters

You can use Identity Manager Debug pages to trace methods in your custom adapter. You must first enable tracing and identify the methods for which tracing is requested. You must also provide calls to create log entries for new methods in your custom adapter.

To debug your adapter, review the log file that is generated by the adapter. If you enabled tracing and identified the methods you wanted to trace, your adapter will write its resource settings to the log file. Use this information to validate that the adapter was started and that all setting changes were saved.

**NOTE** See *Identity Manager Tuning, Troubleshooting, and Error Messages* for detailed information about tracing and debugging custom adapters.

# Maintaining Custom Adapters

Any time you install a new Identity Manager patch or service pack you must test your custom resources with the new `idmcommon.jar` and `idmformui.jar` files. You might have to modify or enhance your adapters so they adapt to changes made in the new release. Alternatively, you might just need to rebuild or refresh your resource adapter in your installation.

When you upgrade to a new release, you might have to recompile all of your custom resource adapters, depending on the target Identity Manager version. All custom Java that uses Identity Manager APIs (including custom resource adapters) require a recompile during upgrading. Also, consider other Java classes that use the Identity Manager library.

For more information about upgrading, see *Identity Manager Upgrade*.

**NOTE** If your current Identity Manager installation has a large amount of custom work, contact your Sun Account Representative or Sun Customer Support for assistance with your upgrade.

Chapter 3

# 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 4

# Using SPML 1.0 with Identity Manager Web Services

Service Provisioning Markup Language (SPML) 1.0 is an OASIS standard used to provide an open interface for communicating with service provisioning activities. You access Identity Manager Web services using SPML requests for HTTP.

This chapter describes SPML 1.0 support in Identity Manager and Identity Manager Service Provider and includes information about 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** Identity Manager supports both SPML version 1.0 and version 2.0.

The concepts in this chapter *relate specifically to SPML 1.0*, but reading this chapter provides a good basis for understanding concepts described in Chapter 5, "Using SPML 2.0 with Identity Manager Web Services."

# Before You Begin

Review the following sections before you start working with Identity Manager Web Services:



## Intended Audience

This chapter is intended for application developers and developers that are responsible for deploying Identity Manager, implementing procedural logic, and using SPML 1.0 classes to format service provisioning request messages and to parse response messages.

## Important Notes

You should be aware of the following information before working with SPML 1.0:

- For optimal performance when you are working with the Identity Manager Web Service Interfaces, use the OpenSPML Toolkit that is co-packaged with Identity Manager. Using the `openspml.jar` file from the `http://www.openspml.org/` web site might cause memory leaks.
- The Service Provider REF Kit contains an `SpmlUsage.java` file that demonstrates how to use the Service Provider SPML interface.
- You can access Identity Manager Service Provider (Service Provider) features through SPML 1.0. (These features are not available with SPML version 2.0.)

  The Service Provider SPML interface is very similar to the Identity Manager SPML interface. Differences in configuration and operation are noted in this chapter where appropriate.

## Related Documentation and Web Sites

In addition to the information provided in this chapter, consult the publications and web sites listed in this section for information related to using SPML.

### Recommended Reading

See Chapter 5, "Using SPML 2.0 with Identity Manager Web Services," in this book for information about using SPML version 2.0.

### Useful Web Sites

Visit the following web site for information about using OpenSPML and to download the OpenSPML 1.0 Toolkit.

`http://www.openspml.org`

# Configuring SPML

To expose the SPML interface, you must properly configure the Identity Manager server by installing and modifying specific repository objects and by editing the `Waveset.properties` file.

Instructions for configuring the SPML interface are provided in the following sections:

- Installing and Modifying Repository Objects
- Editing the Waveset.properties File
- Editing Configuration Objects

# Installing and Modifying Repository Objects

The following table describes the repository objects you must install and modify to configure SPML for Identity Manager.

**Table 4-1** 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. Typically, you define one SPML form for each object class defined in the SPML schema. |
| `Configuration:IDM Schema Configuration` | Defines user attributes that can be stored in the Identity Manager repository for access through an SPML filter, and which are *queryable* and *summary* attributes for Identity Manager user objects.<br>• Define a queryable attribute for attributes you want to use in an SPML filter.<br>• Define a summary attribute for attributes you want returned in an optimized search. |
| `TaskDefinition:SPMLRequest` | System task used to process asynchronous SPML requests.<br>You should not have to customize this object. |

Identity Manager includes a sample set of SPML configuration objects in the `sample/spml.xml` file. You must manually import the `sample/spml.xml` file because the file is not imported by default when the repository is initialized.

The sample configuration defines a `person` class to track the evolving standard schema defined by the SPML working group. *Do not customize this class.* Keep the `person` class consistent with the standard schema, except in the following situation.

When configuring the Service Provider SPML interface, you must install and modify the `Configuration:SPE SPML` configuration object as follows:

- Configure the `person` class (the only objectclass defined by default) to use the Service Provider-specific view handler (`IDMXUser`).
- Use the `form` attribute to define a user form that translates between the SPML request or SPML 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.)

You access the Service Provider SPML interface from the following (default) path:

```
/servlet/spespml
```

For example, if you deploy Identity Manager in the `/idm` context on *`host`*`:`*`port`*, you can access the interface at the following URL:

```
http://host:port/idm/servlet/spespml
```

Where:

- *host* is the machine on which you are running Identity Manager.
- *port* is the number of the TCP port on which the server is listening.

**NOTE** See the SPML 1.0 Specification at `http://www.openspml.org/` for the most current information about the standard SPML schema.

# Editing the `Waveset.properties` File

The following table describes three optional entries in the `Waveset.properties` file that you can use to control how SPML requests are authorized.

**Table 4-2** 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 `soap.epassword` and `soap.password` Properties

A user specified in `soap.username` is known as the *proxy user*.

You can define a proxy user in `soap.username` and specify only one of the following password properties:

- Specifying `soap.password` is the simplest option, but this property exposes a clear text password in the `properties` file.
- Specifying `soap.epassword` is a more secure option, but you must perform extra steps to generate an encrypted password.

Establishing a proxy user is convenient for clients because authentication is not required by the web service. This configuration is common for portal environments where the Identity Manager server is only accessed by other applications that handle user authentication.

**CAUTION** Using a proxy user can be dangerous if the HTTP port on which the responding server resides is generally accessible. Anyone who knows the Identity Manager server's URL, and understands how to build SPML requests, can configure Identity Manager operations for the proxy user to perform.

The SPML standard does not specify how to perform authentication and authorization. Several related web standards are available for authentication, but these standards are not yet in common use. At this time, the most common approach for authentication is to use SSL between applications and the server. Identity Manager does not dictate how to configure SSL.

If you cannot use a proxy user or SSL, Identity Manager supports a vendor-specific extension to SPML that allows the client to log in and maintain a *session token*, which can be used to authenticate subsequent requests. You can use the `LighthouseClient` class (an extension of the `SpmlClient` class that includes support for specifying credentials) to perform a log in request and pass a session token in all SPML requests.

**NOTE** The Service Provider SPML interface does not support authentication and authorization, however you can configure the Identity Manager SPML interface to use the `IDMXUser` view instead of using Service Provider SPML.

Service Provider assumes that clients accessing Identity Manager have been authenticated and authorized by an access management application. The client has all possible rights when using the Service Provider SPML interface.

To prevent sensitive data from being exposed between the client and Identity Manager, consider accessing the Service Provider SPML interface over SSL.

## Creating an Encrypted Password

Use one of the following methods to create an encrypted password:

- Open the Identity Manager console and use the `encrypt` command.
- Open the Identity Manager Debug pages or console and view the XML for the proxy user. Find the `WSUser` element for the `password` attribute value and use that 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 configure the following configuration objects:

- Configuration: SPML Object
- Configuration: SPMLPerson Object
- `Configuration: IDM Schema Configuration` Object
- `TaskDefinition: SPMLRequest` Object
- Deployment Descriptor

**NOTE** The Service Provider SPML interface has only one configuration object, `Configuration:SPE SPML`, which is similar to the `Configuration:SPML` object in structure.

## `Configuration: SPML` Object

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 by using a `GenericObject` that is stored as an extension of the Configuration object.

The following attributes are defined in `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 those classes are mapped onto views. Define one object on this list for each class defined by the SPML schemas on the `schemas` list.

Initially, the distinction between the two lists might be confusing. The information about the **schemas** list defines what Identity Manager returns 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 SPML schemas. Identity Manager works without schemas. If you have not defined an SPML schema, Identity Manager returns an empty response after receiving a schema request message. Without a schema, clients must rely on pre-existing knowledge about the supported classes and attributes.

> ➤ **Best Practice:**
>
> Writing SPML schemas is considered a best practice, so you can use general purpose tools (such as the OpenSPML Browser) to build requests.

### *Default SPML Configuration*

The following example shows the default SPML configuration. The text of the SPML schema definitions have been omitted for brevity.

**Code Example 4-1** Default SPML Configuration

```
<Configuration name='SPML' authType='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>
      <!-- Class 'user' defines no form so we'll default to a builtin simplified schema. I
don't really like this but SimpleRpc currently depends on it.
      -->
      <Object name='user'>
        <Attribute name='type' value='User'/>
        <Attribute name='identifier' value='waveset.accountId'/>
      </Object>
      <!-- Class 'userview' defines the form "view" which causes the view to pass through
unmodified
      -->
      <Object name='userview'>
        <Attribute name='type' value='User'/>
        <Attribute name='form' value='view'/>
        <Attribute name='identifier' value='waveset.accountId'/>
        <Attribute name='multiValuedAttributes'>
          <List>
            <String>waveset.resources</String>
            <String>waveset.roles</String>
```

**Code Example 4-1** Default SPML Configuration *(Continued)*

```
            <String>waveset.applications</String>
          </List>
        </Attribute>
      </Object>

      <Object name='role'>
        <Attribute name='type' value='Role'/>
        <Attribute name='form' value='SPMLRole'/>
        <Attribute name='default' value='true'/>
        <Attribute name='identifier' value='name'/> <!--  attribute ...for now? -->
      </Object>
</Configuration>

</Waveset>
```

Two classes are defined in this example:

- The standard `person`
- An Identity Manager extension named `request`

The following attributes are supported in a class definition:

- **name**: Identifies the name of the class. The `name` value can correspond to an `<ObjectClassDefinition>` element in an SPML schema, although this value is not required. You can use this name as the value for the `objectclass` attribute in an Add request or Search request.
- **type**: Defines the Identity Manager view type used to manage instances of this class. Generally, this attribute is `User`, but can be any repository type that is accessible 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 attribute contains the rules for transforming between the external attributes defined by the class and the internal view attributes.
- **default**: Specify `true` to indicate that this attribute is the default class for this type only. For more than one SPML class implemented on the same type, you must designate one class 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 evaluating an SPML search request for a class, you typically include all repository objects associated with that class in that search. This approach is fine for `User` objects, but some classes might be implemented by 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 typically created 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, note that the schema elements are surrounded by a `CDATA`-marked section. Using `CDATA`-marked sections is convenient for escaping long strings of XML. When Identity Manager normalizes the `spml.xml` file, the `CDATA`-marked sections are converted into strings containing `&lt;` and `&gt;` 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** and various extended requests for common account management operations.

## `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 following example shows how the standard `person` class references a form.

**Code Example 4-2** 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>
         <Field name='gn'>
            <Derivation><ref>global.firstname</ref></Derivation>
         </Field>
         <Field name='global.firstname'>
            <Expansion><ref>gn</ref></Expansion>
         </Field>
```

**Code Example 4-2** Standard Person Class References Form *(Continued)*

```
         <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
- Are defined only for data transformation
- Are not intended for interactive editing

For each 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.

### `Configuration: IDM Schema Configuration` Object

If you want to use attributes in an SPML search filter, you must define those attributes as *extended attributes* for Identity Manager users. Identity Manager stores extended attribute values in the repository, even when that value is also stored as a resource account attribute.

Try to minimize the number of extended attributes. Too many extended attributes can increase the repository size and might cause consistency problems between attributes stored in Identity Manager and the real value of the attribute stored on a resource. To use an attribute in an Identity Manager query, the attribute must be declared as extended so that the value is accessible when the repository query indexes are built.

If you want to include attributes in a user's set of *summary attributes*, you must define those attributes as *extended attributes*. 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 user attributes. In the Identity Manager SPML implementation, summary attributes are returned when you do not explicitly provide a list of return attributes in the search request.

In the following example, `firstname`, `lastname`, `fullname`, `description`, and `telephone` are extended attributes that are present on the User `IDMObjectClassConfiguration` after being defined in the `IDMAttributeConfigurations`. Only `firstname`, `lastname`, and `telephone` are queryable *and* summary attributes.

**Code Example 4-3** `telephone` and `description` Declared as Extended Attributes

```
<Configuration name="IDM Schema Configuration"
                 id='#ID#Configuration:IDM_Schema_Configuration'
                 authType='IDMSchemaConfig'>
       <IDMSchemaConfiguration>
           <IDMAttributeConfigurations>
               <!-- this is the standard set -->
               <IDMAttributeConfiguration name='firstname'
                                          syntax='STRING'/>
               <IDMAttributeConfiguration name='lastname'
                                          syntax='STRING'/>
               <IDMAttributeConfiguration name='fullname'
                                          syntax='STRING'/>
               <!-- these are the SPML extensions -->
               <IDMAttributeConfiguration name='description'
                                          syntax='STRING'/>
               <IDMAttributeConfiguration name='telephone'
                                          syntax='STRING'/>
           </IDMAttributeConfigurations>
           <IDMObjectClassConfigurations>
               <IDMObjectClassConfiguration name='User'
                                            extends='Principal'
                                            description='User description'>
                   <IDMObjectClassAttributeConfiguration name='firstname'
                                                         queryable='true'
                                                         summary='true'/>
                   <IDMObjectClassAttributeConfiguration name='lastname'
                                                         queryable='true'
                                                         summary='true'/>
```

**Code Example 4-3** `telephone` and `description` Declared as Extended Attributes

```
                <IDMObjectClassAttributeConfiguration name='fullname'/>
                <IDMObjectClassAttributeConfiguration name='description'/>
                <IDMObjectClassAttributeConfiguration name='telephone'
                                                      queryable='true'
                                                      summary='true'/>
            </IDMObjectClassConfiguration>
        </IDMObjectClassConfigurations>
    </IDMSchemaConfiguration>
</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 same `Configuration:` 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).

### `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, it launches a new instance of this task and passes the SPML message as an input variable for the task. The server then returns the task instance repository ID 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. However, you might want to change the `resultLimit` value. When asynchronous requests have completed, the system typically retains the result value for a specified time so the client can issue an SPML status request to obtain the results. How long to retain these results is site-specific.

Use a positive `resultLimit` value to specify how long (in seconds) the system can retain results after completing a task. The default value for `SPMLRequests` is typically 3600 seconds, or approximately one hour. Other tasks default to 0 seconds unless you change the task name to a different value.

If negative, the request instance is never removed automatically.

**TIP** Set the value of `resultLimit` to the shortest possible time to avoid cluttering the repository.

**NOTE** The Service Provider SPML interface does not support asynchronous requests.

## Deployment Descriptor

You must edit the Identity Manager deployment descriptor, typically found in the file `WEB-INF/web.xml`, to contain a declaration for the servlet that receives SPML requests.

If you are having difficulty contacting the SPML web service, look in the `web.xml` file for a servlet declaration. The following example shows a servlet declaration.

**Code Example 4-4** 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`

Where

- *host* is the machine on which you are running Identity Manager.
- *port* is the number of the TCP port on which the server is listening.

Although you can, you are not required to define a *<servlet-mapping>*. Do not modify the contents of this servlet declaration.

# Starting the SPML Browser

You can use the OpenSPML Browser application to test the Identity Manager SPML configuration.

To start the browser,

1. Open a command window.
2. At the command prompt, type the following command:

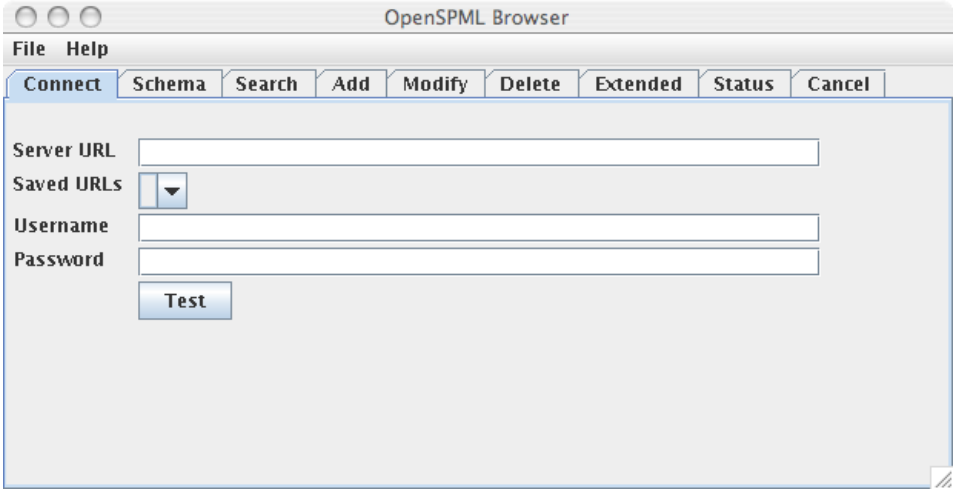
   ```
   lh spml
   ```

# Connecting to the Identity Manager Server

To connect to the Identity Manager server,

1. Open the OpenSPML browser and select the **Connect** tab.

**Figure 4-1** Example OpenSPML Browser




2. Type the URL of the Identity Manager server.

   For example, if the server is running on port 8080 on a local machine, the URL would be:

   ```
   http://host:8080/idm/servlet/rpcrouter2
   ```

# Testing and Troubleshooting Your SPML Configuration

To test your SPML configuration:

1. Select the Connect tab and click Test.

   A dialog displays to indicate the connection was successful.

2. Select the Schema tab 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

- Verify that you typed the URL correctly.
- 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

After configuring the server, your SPML application requires a mechanism for sending SPML messages and receiving SPML responses. For Java applications, use the OpenSPML Toolkit to configure this mechanism.

**NOTE** For optimal performance when you are working with the Identity Manager Web Service Interfaces, use the OpenSPML Toolkit that is co-packaged with Identity Manager.

Using the `openspml.jar` file from the `http://www.openspml.org/` web site might cause memory leaks.

The toolkit can provide 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

The following table describes the most important classes provided by the OpenSPML Toolkit. Each request type has a corresponding class. Consult the JavaDocs distributed with the toolkit for complete information.

**Table 4-3** Classes Provided by OpenSPML Toolkit

| Class | Description |
|---|---|
| `AddRequest` | Constructs a message to request creation of a new object.<br>You define the object type by passing an `objectclass` attribute. Other passed attributes must adhere to the schema associated with the object class. SPML does not yet define standard schemas, but you can configure Identity Manager to support almost all schemas. |
| `BatchRequest` | Constructs a message that can contain more than one SPML request. |
| `CancelRequest` | Constructs a message to cancel a request that was formerly executed asynchronously. |
| `DeleteRequest` | Constructs a message to request the deletion of an object. |
| `ModifyRequest` | Constructs a message to request modification of an object. Include only those attributes that you want to modify in the request. Attributes not included in the request retain their current value. |
| `SchemaRequest` | Constructs a message to request information about SPML object classes supported by the server. |
| `SearchRequest` | Constructs a message to request object attributes that match certain criteria. |
| `SpmlClient` | Presents a simple interface for sending and receiving SPML messages. |
| `SpmlResponse` | Includes the base class for objects representing response messages sent back from the server. Each request class has a corresponding response class. For example, `AddResponse` and `ModifyResponse`. |
| `StatusRequest` | Constructs a message to request the status of a request that was formerly executed asynchronously. |

The Service Provider REF Kit contains an `SpmlUsage.java` file that demonstrates how to use the Service Provider 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 Service Provider SPML interface and defaults to

`http://`*`host`*`:`*`port`*`/idm/spespml`

Where

- *host* is the machine on which you are running Identity Manager Service Provider.
- *port* is the number of the TCP port on which the server is listening.

You can enable trace for Service Provider to print Service Provider SPML messages to standard output.

## `ExtendedRequest` Examples

The following table describes the different `ExtendedRequest` classes you can use to send messages to and receive messages from the client.

**Table 4-4** `ExtendedRequest` Classes for Sending and Receiving Messages

| ExtendedRequest | Description |
|---|---|
| `changeUserPassword` | Constructs a message to request a user password change. |
| `deleteUser` | Constructs a message to request the deletion of a user. |
| `disableUser` | Constructs a message to request the disabling of a user. |
| `enableUser` | Constructs a message to request the enabling of a user. |
| `launchProcess` | Constructs a message to request 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. |
| `resetUserPassword` | Constructs a message to request the reset of a user password. |
| `runForm` | Allows you to create custom SPML requests that return information obtained by calling the Identity Manager Session API. |

The server code converts the `ExtendedRequests` into view operations.

Examples using the typical formats for these classes are presented in the following sections:



## ExtendedRequest Example

The following example shows the typical format for an ExtendedRequest.

**Code Example 4-5** ExtendedRequest Format

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("changeUserPassword");
req.setAttribute("accountId", "exampleuser");
req.setAttribute("password", "xyzzy");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

Most SPML ExtendedRequests accept the following arguments:

- accountId: Identifies the Identity Manager user name
- accounts: Presents resource names in a comma-delimited list

If you do not pass an accounts attribute, the operation updates all resource accounts linked to the user, including the Identity Manager user account. If you do pass accounts, the specified SPML operation only updates 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 Example

The following example shows the typical format for a deleteUser request (View > Deprovision view).

**NOTE** If you customize this request, there might be side effects.

**Code Example 4-6** deleteUser Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("deleteUser");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

## disableUser Example

The following example shows the typical format for a disableUser request (View > Disable view).

**Code Example 4-7** disableUser Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("disableUser");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

## enableUser Example

The following example shows the typical format for an enableUser request (View > Enable view).

**Code Example 4-8** enableUser Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("enableUser");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

## `launchProcess` Example

The following example shows the typical format for a `launchProcess` request. (View > Process view).

**Code Example 4-9** `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:

- `launchProcess`: Starts custom processes.
- `process` : Name of the `TaskDefinition` object in the Identity Manager repository to start.
- `taskName`: Name of the task needed to start the workflow.

  The `task instance` object holds the runtime state of the process.

The remaining attributes are arbitrary and they are passed into the task.

## `listResourceObjects` Example

The following example shows the typical format for a `listResourceObjects` request.

**Code Example 4-10** `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

### `resetUserPassword` Example

The following example shows the typical format for a `resetUserPassword` request (View > Reset User Password view).

**Code Example 4-11** `resetUserPassword` Request

```
ExtendedRequest req = new ExtendedRequest();
req.setOperationIdentifier("resetUserPassword");
req.setAttribute("accountId","exampleuser");
req.setAttribute("accounts","Lighthouse,LDAP,RACF");
ExtendedResponse res = (ExtendedResponse) client.send(req);
```

### `runForm` Example

The following example shows the typical format for a `runForm` request.

**Code Example 4-12** `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 following example shows a form that runs queries and returns a list of the Role, Resource, and Organization names accessible to the current user.

**Code Example 4-13** Query Form

```
<Configuration name='SPML Get Object Names'>
  <Extension>
    <Form>
      <Field name='roles'>
        <Derivation>
          <invoke class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>Role</s>
          </invoke>
        </Derivation>
      </Field>
      <Field name='resources'>
        <Derivation>
          <invoke class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>Resource</s>
          </invoke>
        </Derivation>
      </Field>
      <Field name='organizations'>
        <Derivation>
          <invoke class='com.waveset.ui.FormUtil'>
            <ref>display.session</ref>
            <s>ObjectGroup</s>
          </invoke>
        </Derivation>
      </Field>
    </Form>
  </Extension>
</Configuration>
```

You use the `runForm` request 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 and use a `searchRequest` to query for their names. However, this configuration requires four `searchRequests` to gather the information. To reduce the number of SPML requests, encode the queries in a form by using a single `runForm` request to perform the queries, and returning the combined results.

## Using Trace with SPML

SPML includes options for turning on trace output so you can log Identity Manager's SPML traffic and diagnose problems.

For more information about tracing SPML, see the "Tracing and Troubleshooting Identity Manager" chapter in the *Identity Manager Tuning, Troubleshooting, and Error Messages* book.

# Example Methods for Implementing SPML

This section presents examples that show several common methods for implementing SPML:

- Add Request
- Modify Request
- Search Request

## Add Request

An example Add Request is shown in Code Example 4-14:

**Code Example 4-14** Add Request

```
SpmlClient client = new SpmlClient();
   client.setURL("http://example.com:8080/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 contains two, example Authenticated SPML Modify Requests.

**Code Example 4-15** Authenticated SPML Request

```
SpmlClient client = new SpmlClient();
   client.setURL("http://example.com:8080/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 4-16** Authenticated SPML Request with LighthouseClient

```
LighthouseClient client = new LighthouseClient();
   client.setURL("http://example.com:8080/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");
```

The only difference between these examples is that the Code Example 4-16 uses the `LighthouseClient` class and two additional method calls to `client.setUser` and `client.setPassword`. For example, you could use this example to avoid setting a proxy user in `Waveset.properties`, which results in the audit log reflecting the specified user instead of the proxy user.

This example is authenticated by `client.setUser` and `client.setPassword` when the request is sent.

# Search Request

An example Search Request is shown in Code Example 4-17:

**Code Example 4-17** Search Request

```
SpmlClient client = new SpmlClient();
   client.setURL("http://example.com:8080/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);
   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 5

# Using SPML 2.0 with Identity Manager Web Services

This chapter describes SPML 2.0 support in Identity Manager 8.0; 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 4, "Using SPML 1.0 with Identity Manager Web Services," which also contains useful information about using SPML.

This information is organized as follows:

- Before You Begin
- Overview
- Configuring Identity Manager to Use SPML 2.0
- Extending the System
- Sample SPML 2.0 Adapter

# Before You Begin

Review the following sections before you start working with Identity Manager Web Services:



## Intended Audience

This chapter is intended for application developers and developers who are responsible for deploying Identity Manager, implementing procedural logic, and using SPML 2.0 classes to format service provisioning request messages and to parse response messages.

## Important Notes

You should be aware of the following information before working with SPML 2.0:

- For best performance when working with the Identity Manager Web Service Interfaces, use the OpenSPML Toolkit supplied with Identity Manager. Using the `openspml.jar` file from the `http://www.openspml.org/` web site might cause memory leaks.
- When implementing SPML 2.0, you must modify the configuration to add the `spml2ObjectClass` attribute to your schema. The `objectclass` attribute value provided in previous releases is now maintained in the `spml2ObjectClass` attribute.
- You cannot access Identity Manager Service Provider (Service Provider) features through SPML 2.0. (These features are available with SPML version 1.0.)

## Related Documentation and Web Sites

In addition to the information provided in this chapter, consult the publications and web sites listed in this section for information related to using SPML.

### Recommended Reading

See Chapter 4, "Using SPML 1.0 with Identity Manager Web Services," in this book for information about using SPML version 1.0.

### Useful Web Sites

Visit the following web site for information about using OpenSPML, to read SPML 2.0 Specifications, and to download the OpenSPML 2.0 Toolkit.

`http://www.openspml.org`

# Overview

This section explains some basic concepts about SPML 2.0:

- How SPML 2.0 Compares to SPML 1.0
- How SPML 2.0 Concepts Are Mapped to Identity Manager
- Supported SPML 2.0 Capabilities

## 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 4, "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.
- 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 (see Table 5-1) that will make it possible for you to extend capabilities or add new capabilities in the future.

**Table 5-1** 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:<br>• Async: Asynchronous processing of requirements<br>• Batch: Process a batch of requests<br>• Bulk: Process modifies or deletes using iteration<br>• Password: Change, set, reset, validate, or expire passwords<br>• Reference: Refer to PSOs between targets<br>• Suspend: Enable or disable PSOs<br>• 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.

**NOTE** See the OASIS SPML 2.0 Specifications at `http://www.openspml.org/`.

This section describes how the following SPML 2.0 concepts are mapped into Identity Manager:

- Target
- PSO
- PSOIdentifier
- Open Content and Operational Attributes

## 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.

A supported target is the one target defined in the `spml2.xml` file (`Configuration:SPML2` object). For example, in Code Example 5-6 on page 232 `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** Identity Manager only manages Users and requires you to define a user extended attribute called `spml2ObjectClass`.

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` that is provided to support additional targets later.
- A form that maps the attributes to and from the Identity Manager view.

## PSOIdentifier

SPML includes an object ID that is called a *PsoID*.

OASIS SPML 2.0 Specifications recommend that PSOIdentifiers (PsoID) should be opaque to a requestor (client). Consequently, Identity Manager uses repository IDs (repoIDs) as the PsoID when adding PSOs to the system.

The repoID is distinct and it 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. If this attribute is not there when a request is made, Identity Manager allows you to specify and use a "default" objectclass, such as `SPMLUser`. Internally, the objectclass value is maintained as an `spml2ObjectClass` attribute for users. For Identity Manager this attribute must be a user extended attribute. You might not see an `spml2ObjectClass` 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 supports all Core capabilities in the SPML 2.0 Specification that use 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, where Identity Manager (knowingly) varies from the specification and profile document, and which operational attributes are required by Identity Manager.

This information is organized into the following sections:

- Core Capabilities
- Async Capabilities
- Batch Capability
- Bulk Capabilities
- Password Capabilities
- Suspend Capabilities

**NOTE** Identity Manager does not support the Reference capability, the Search capability, the Updates capability, or the `CapabilityData` class.

None of the supported capabilities use the `CapabilityData` class, so Identity Manager does not support it. (The `CapabilityData` class is used to implement custom capabilities.)

The OpenSPML 2.0 Toolkit supports `CapabilityData` in the marshallers, unmarshallers, and so forth.

## Core Capabilities

Identity Manager supports the following Core capabilities:

**Table 5-2** Core Capabilities

| Capability | Description | Caveats |
|---|---|---|
| AddRequest | Adds a specified PSO to the system. | Identity Manager officially supports only a single target. |
| DeleteRequest | Deletes a specified PSO from the system. | Identity Manager officially supports only a single target. |
| ListTargetsRequest | Lists the targets that are available through Identity Manager. | • Identity Manager officially supports a single target.<br>• 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`.)<br><br>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 |
| ModifyRequest | Modifies specified PSO attributes. | 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:

- You can provide `username` and `password` values for the `ListTargetsRequest` request. These values are used as credentials to establish a `session`, which is identified by the `session token` value returned in the`ListTargetsRequest` response. This session is the context for all following requests that include that `session token` value as an operational attribute.

  Another way to set up the session is to provide values for the `soap.username` and `soap.password` attributes in `Waveset.properties`. In this case, no `session token` is required.

- Identity Manager supports only the DSML Profile.

`AddRequest` and `ListTargetRequest` examples follow.

### *AddRequest Examples*

This section provides several AddRequest examples.

The following example shows a `.jsp` that invokes a `ListTargetsRequest` through Identity Manager's `SessionAwareSpml2Client` class.

**Code Example 5-1** 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://host:port/idm/servlet/openspml2";
%>

<html>
<head><title>SPML2 Test</title></head>
<body>
<%

 // need a client.
 SessionAwareSpml2Client client = new SessionAwareSpml2Client( url );

 // login
 client.login("configurator", "password");
```

**Code Example 5-1** Example Client Code *(Continued)*

```
// AddRequest
String rid = "rid-spmlv2"; // The RequestId is not strictly required.

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 5-2 shows the SPML 2.0 request that was sent.

**Code Example 5-2** 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 5-3 shows the body of the SPML request that is returned to the client.

**Code Example 5-3** 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 5-4 shows a `.jsp` invokes a `ListTargetsRequest` via Identity Manager's `SessionAwareSpml2Client` class.

**Code Example 5-4** 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://host:port/idm/servlet/openspml2";
%>
```

**Code Example 5-4** 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 5-5 shows the body of the SPML request that is sent.

**Code Example 5-5** 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 5-6 shows the body of the SPML request that is received by or returned to the client.

**Code Example 5-6** 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 5-6** 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 Capabilities

Identity Manager supports the Async capabilities described in Table 5-3:

**Table 5-3** 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

Identity Manager supports the Batch capability described in Table 5-4.

**Table 5-4** Batch Capability

| Capability | Description | Operational Attributes | Caveats |
|---|---|---|---|
| BatchRequest | Executes a batch of requests. | None | |

## Bulk Capabilities

Identity Manager supports the Bulk capabilities described in Table 5-5:

**Table 5-5** 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 Capabilities

Identity Manager supports the Password capabilities described in Table 5-6:

**Table 5-6** 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.<br>• Identity Manager does not support the `remainingLogins` attribute.<br><br>If you set this attribute to anything other than the default (1 or less), an `OperationNotSupported` error occurs. |
| ResetPasswordRequest | Resets the password and returns the new value on all accounts. | None | Passwords are sensitive. Use SSL or some other secure transport. |

**Table 5-6** Password Capabilities *(Continued)*

| Capability | Description | Operational Attributes | Caveats |
|---|---|---|---|
| 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 5-7 is an example ResetPasswordRequest.

**Code Example 5-7** Example ResetPasswordRequest

```
ResetPasswordRequest rpr = new ResetPasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
rpr.setPsoID(psoId);
...
```

### *SetPasswordRequest Example*

Code Example 5-8 is an example SetPasswordRequest.

**Code Example 5-8** 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 5-9 is an example ValidatePasswordRequest.

**Code Example 5-9** Example ValidatePasswordRequest

```
ValidatePasswordRequest vpr = new ValidatePasswordRequest();
...
PSOIdentifier psoId = new PSOIdentifier(accountId, null, null);
vpr.setPsoID(psoId);
vpr.setPassword("apassword");
...
```

## Suspend Capabilities

Identity Manager supports the Suspend capabilities described in Table 5-7.

**Table 5-7** Suspend Capabilities

| Capability | Description | Operational Attributes | Caveats |
|---|---|---|---|
| ResumeRequest | Resumes (enables) a PSO User. | None | Does not support `EffectiveDate`.<br>If you set `EffectiveDate`, Identity Manager returns an `OperationNotSupported` error. |
| SuspendRequest | Suspends an accounts/PSO (disable) | None | Does not support `EffectiveDate`.<br>If you set `EffectiveDate`, Identity Manager returns an `OperationNotSupported` error. |

# Configuring Identity Manager to Use SPML 2.0

This section describes how to configure to use Identity Manager to use SPML 2.0. The topics include:



## Deciding Which Attributes to Manage

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`

You must map these attributes to the User view.

This section also provides short examples that demonstrate how to manage PSOs using SPML 2.0 support in Identity Manager.

Identity Manager provides a sample set of SPML configuration objects in the `sample/spml2.xml` file. You must manually import the `sample/spml2.xml` file because it is not imported by default when the repository is initialized. See the contents of this file for detailed information.

**NOTE** The `spml2ObjectClass` attribute is not present in the `User` schema by default. If this attribute is not already enabled, you must manually add the `spml2ObjectClass` attribute to your schema before Identity Manager can function as an SPML 2.0 server.

The `spml2ObjectClass` attribute has been defined in the `schema.xml` supplied with Identity Manager, but the section where you add this attribute to the configuration is commented out. Assuming that your production schema is in a file derived from that original, you can uncomment that section, import or re-import the schema file, and restart Identity Manager to enable use of the SPML 2.0 feature.

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.

# Configuring the SPML2 Configuration Object

The `sample/spml2.xml` file contains an out-of-the-box configuration for SPML 2.0 support. You can import this file, or one derived from this file, to define the objects that Identity Manager needs to support SPML 2.0.

You can use the `SPML2` configuration type object to change how SPML 2.0 support behaves or to extend the system.

**NOTE** See "Extending the System" on page 241 for more information about extensions.

# Configuring web.xml

If you are using a servlet container like Tomcat, you use web.xml to set up the openspmlRouter servlet, which is the servlet that handles SPML 2.0 requests.

**NOTE** The web.xml file ships with a default installation and no action is required for this component.

The web.xml file contains an optional init-param that you can use to open a monitor window (in Swing) that displays the flow of SPML 2.0 messages. You can use this window to monitor the flow of SPML 2.0 messages, which is useful for debugging purposes.

The following example shows how to add the init-param.

```
<init-param>
    <param-name>monitor</param-name>
 <param-value>org.openspml.v2.util.SwingRPCRouterMonitor</param-value>
</init-param>
```

The following example contains a commented section and contains information about other init-params.

**Code Example 5-10** 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>
```

**Code Example 5-10** Commented Example*(Continued)*

```
    <!--
        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>
    </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>
```

## Configuring SPML Tracing

SPML provides options for turning on trace output so you can log Identity Manager's SPML traffic and diagnose problems.

For more information about tracing SPML, see the "Tracing and Troubleshooting Identity Manager" chapter in the *Identity Manager Tuning, Troubleshooting, and Error Messages* book.

# Extending the System

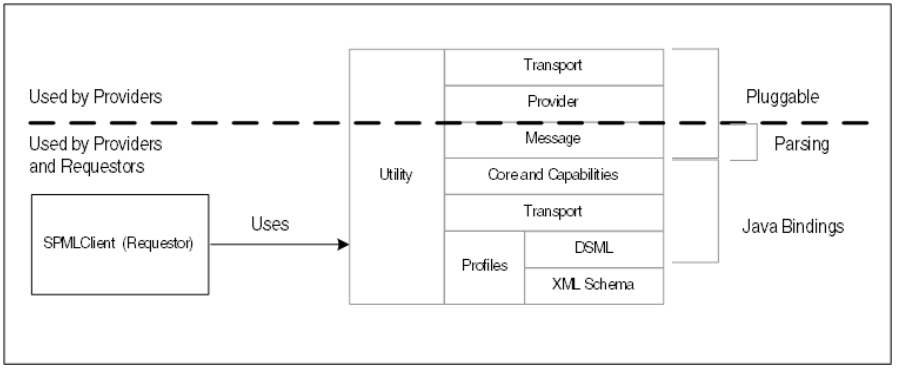
You extend the schema by modifying the configuration object, and you can add executors for requests by changing the section. Using forms, you can map DSML to Views and back.

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. The following figure shows the OpenSPML 2.0 Toolkit architecture.

**Figure 5-1** OpenSPML 2.0 Toolkit Architecture



# Sample SPML 2.0 Adapter

Identity Manager provides a sample SPML 2.0 resource adapter. Using this adapter as a starting point, you can modify the content to communicate with Identity Manager installations or third-party resources that support SPML 2.0 core operations.

**NOTE** You will find this sample adapter in the Sun Resource Extension Facility Kit on your product CD or in your install image located in `/REF`.

Appendix A

# Using the Business Process Editor

**NOTE** The Business Process Editor (BPE) is deprecated, and will be removed in the next Identity Manager release. Please use the Identity Manager IDE instead.

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 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

To start the BPE from the command line:

1. Change to the Identity Manager installation directory.
2. Set environment variables with these commands:

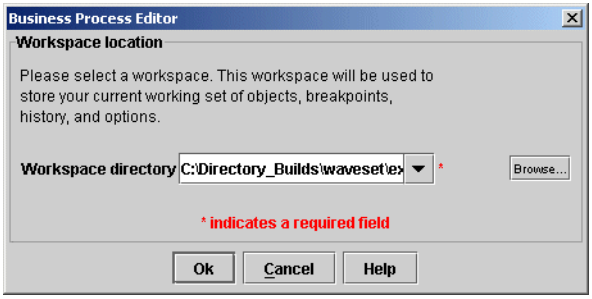
   ```
   set WSHOME=<Path_to_idm_directory>
   set JAVA_HOME=<path_to_jdk>
   ```

   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:

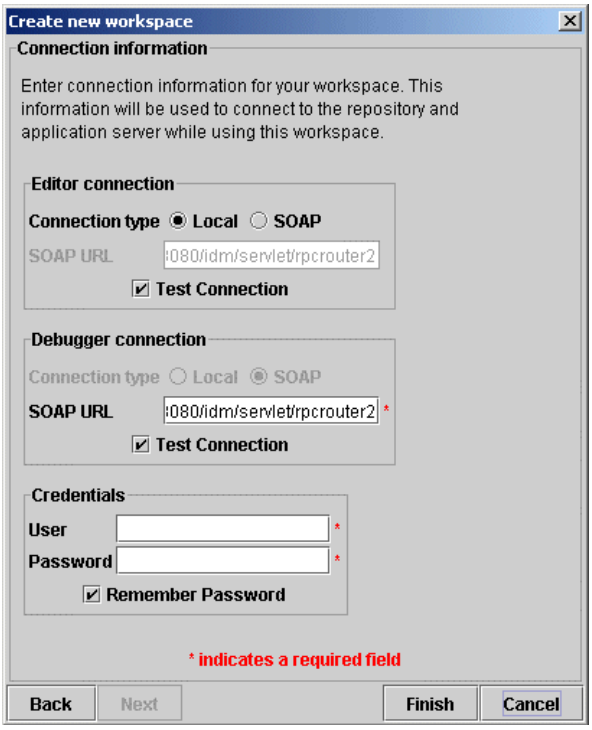
1. In the Workspace location dialog, enter a unique name for the new workspace in the Workspace Directory field, and then click OK.

   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://*host*:*port*/*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 must be the same as the URL that you entered as the debugger connection. For example, http://*host*:*port*/idm.

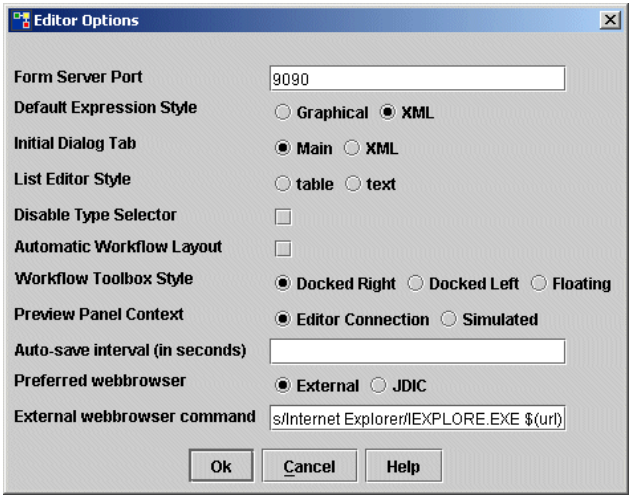
## Enabling JDIC

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

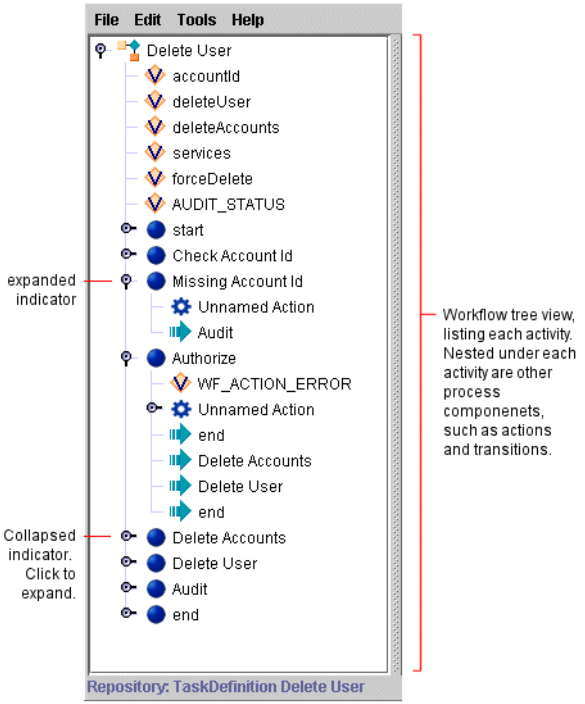
The BPE interface includes a menu bar and dialogs for selections. The primary display is divided into two main panes:

- 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.

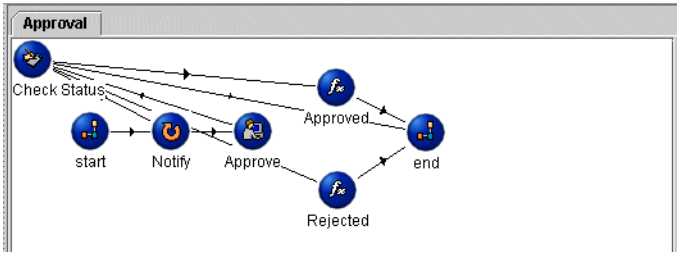
These views are introduced in the following sections.

**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)



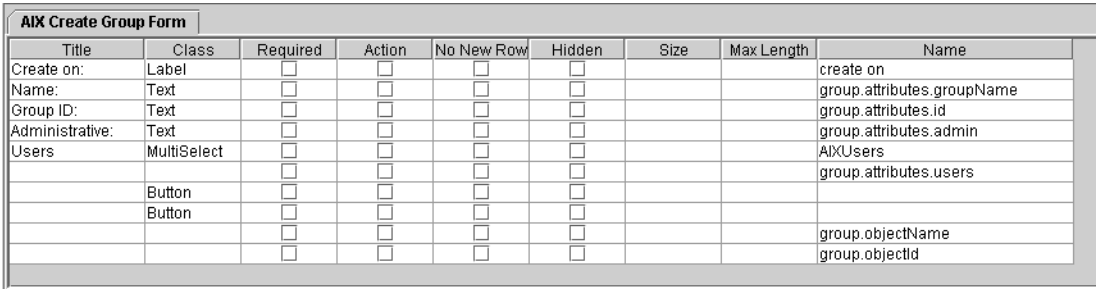

### *Graphical View*

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 260, 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 and it contains a list of objects, which can include the following object types.

   - Workflow Processes
   - Libraries
   - Workflow Sub-processes
   - Generic Objects
   - Forms
   - Configuration Objects
   - Rules
   - Email Templates

   Items displayed may vary based on your Identity Manager implementation.

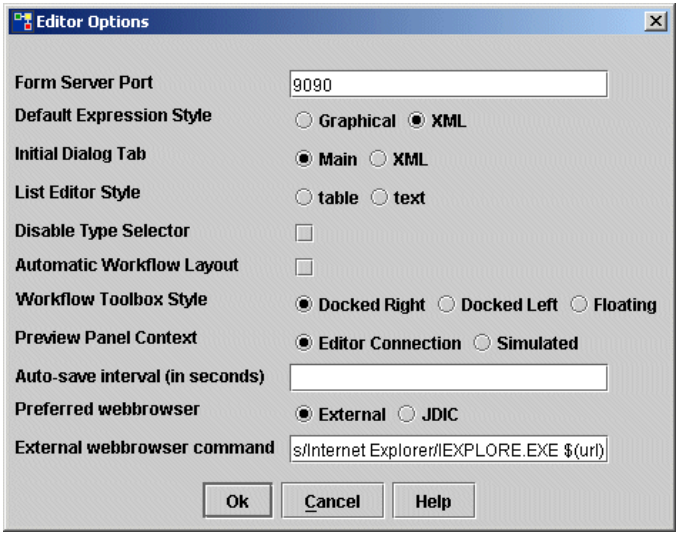
3. Double-click an object type to display all of the objects that you have permission to view for that type.
4. Select a process or object, and then click OK.

# 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-7** 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 try and 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`. |
|---|---|

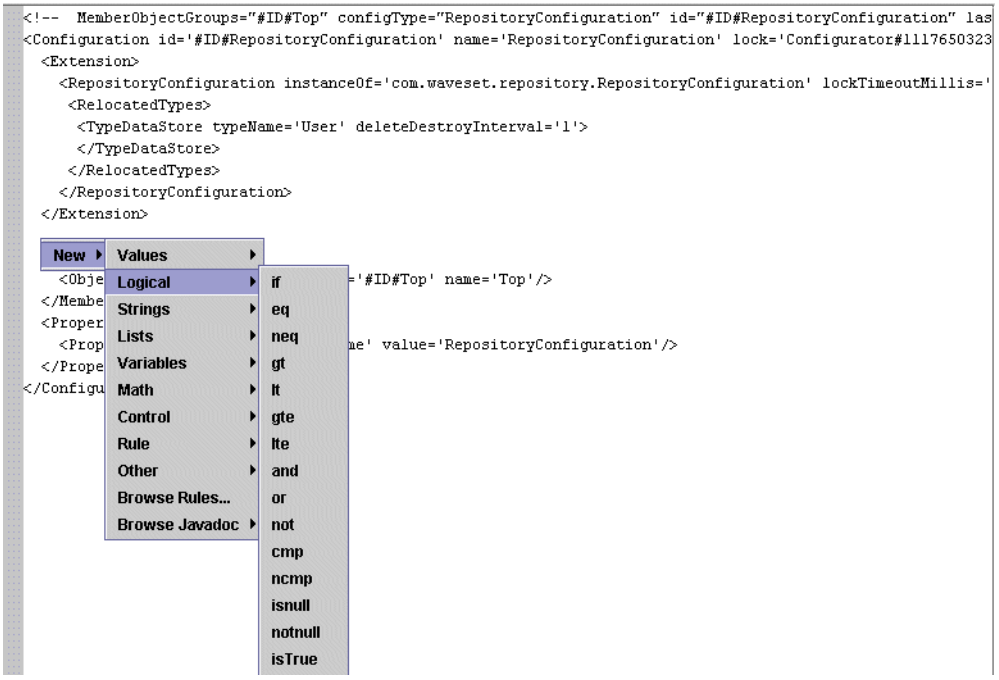
# Inserting XPRESS

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.

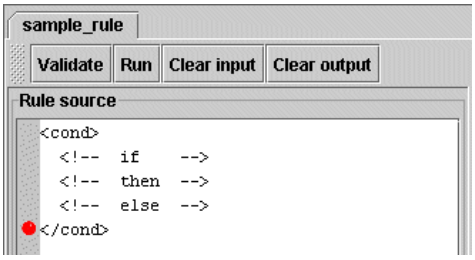
**Figure A-8** Menu for Inserting XPRESS Functions into XML




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-9** Inserting XPRESS Function




# Using Keyboard Shortcuts

The BPE supports these keyboard shortcuts for performing tasks.

**Table A-1** BPE Keyboard Shortcuts

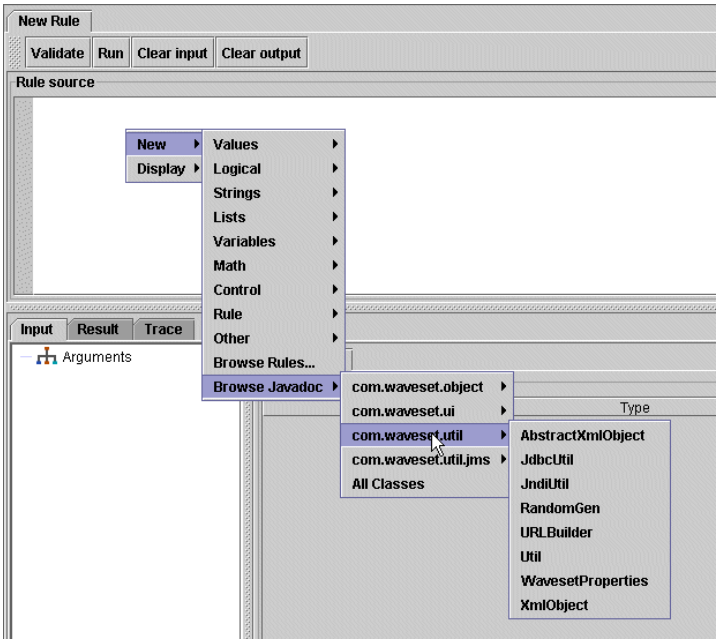
| Keyboard Command/Key | Action |
|---|---|
| Ctrl-C | Copy |
| Ctrl-O | Open (repository object) |
| Ctrl-R | Refresh Sources |
| Ctrl-S | Save (repository object) |
| Ctrl-V | Paste |
| Ctrl-X | Cut |
| Delete | Delete |
| F5 | Select Current Line |
| F6 | Step out |
| F7 | Step into |
| F8 | Step 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-10** 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.

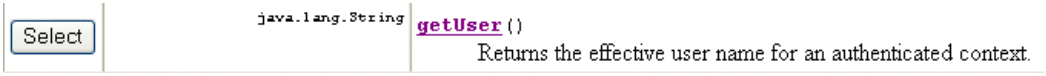
- **All Classes**: Displays the frames view of the JavaDoc classes, from which you can navigate through each class in the browser.

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-11** 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.

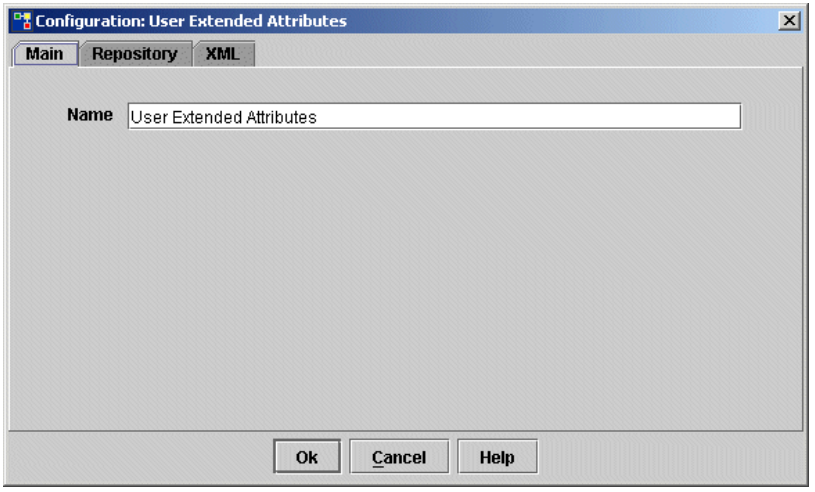
The BPE lists these miscellaneous configuration objects in the left pane (tree view), as shown in the following figure:

**Figure A-12** 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.

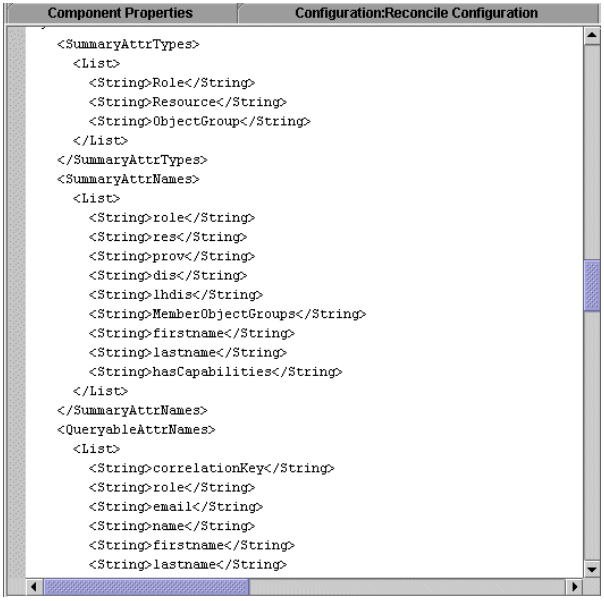
For example, if you double-click User Extended Attributes in tree view, the following dialog displays:

**Figure A-13** 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:

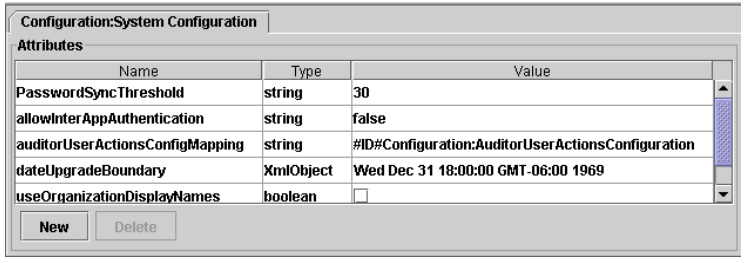
**Figure A-14** BPE XML Display of Reconcile Configuration Object




## 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-15** 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.

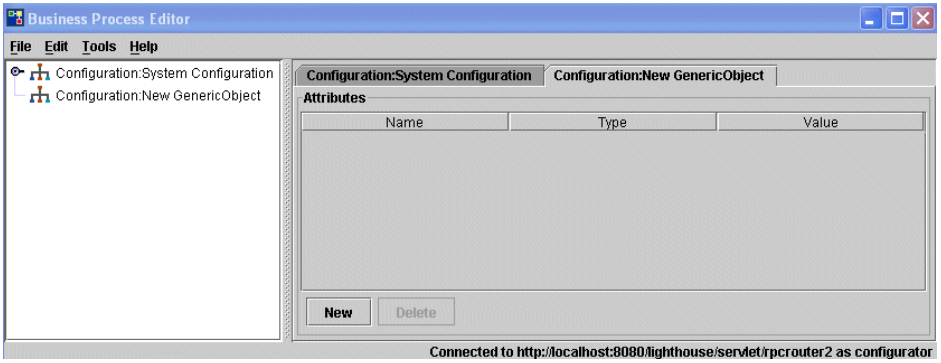
   The Configuration:New GenericObject or Configuration:New Configuration dialog opens, with the Main panel displayed.

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,

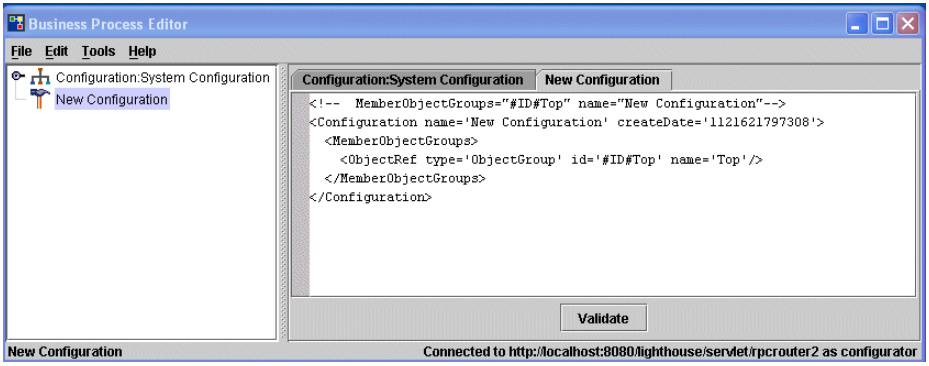
   - If you created a generic object, a blank Attributes pane displays as follows:

**Figure A-16** 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-17** BPE New Configuration Object Display

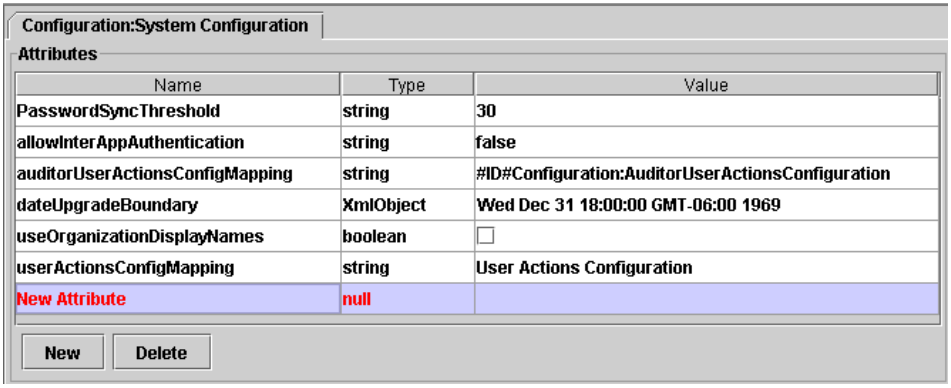


3. If you are creating a generic object, add an attribute as follows, repeating as necessary:

   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-18** 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 244.

## 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

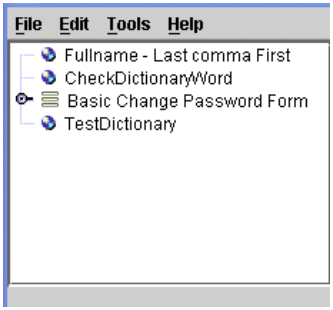
When you are working with rules, the BPE interface provides the following display panes:

- 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-19** Rule Display in Tree View



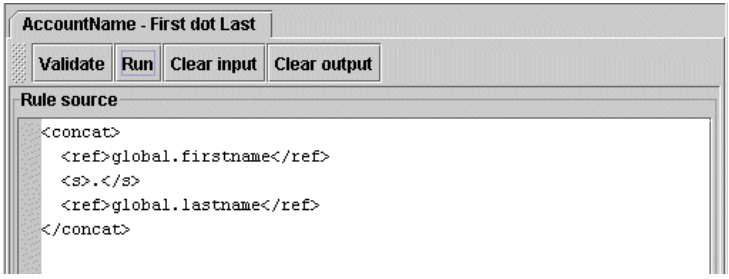

Typically, the tree view shows a hierarchical view of tasks, forms, or views — listing each element in order with sub-elements nested under their parent.

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-20** 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.

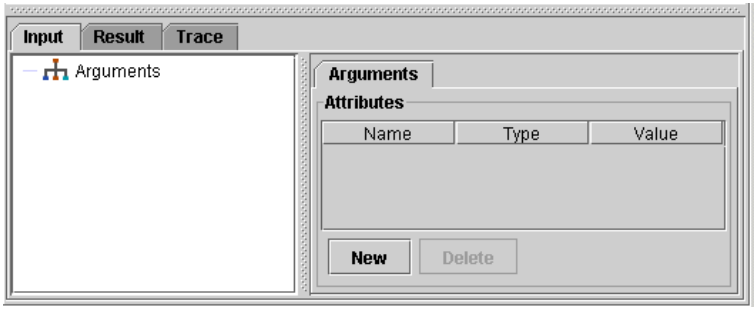
You can also use the buttons located above this pane to perform any of the following actions:

- **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-21** 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 265).

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*

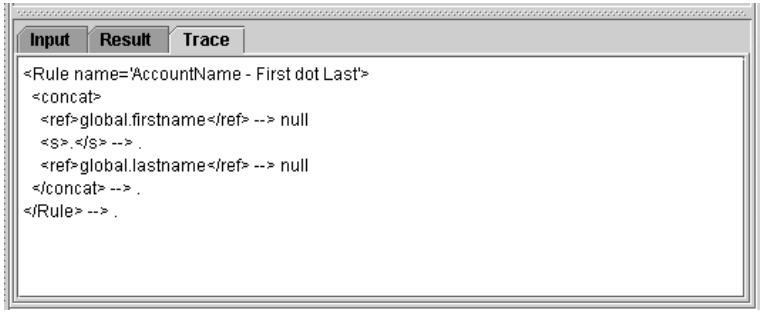
Select the Result tab and click Run (above the Rule source pane) to execute the selected rule. The rule's return value displays in the Result tab pane in XML format.

**Figure A-22** Result Tab Pane


### *Trace Tab*

Select the Trace tab to capture XPRESS tracing during execution of the rule.

**Figure A-23** Trace Tab Pane




## Menu Selections

You can use the menu bar or the action (right-click) menu to work in the interface.

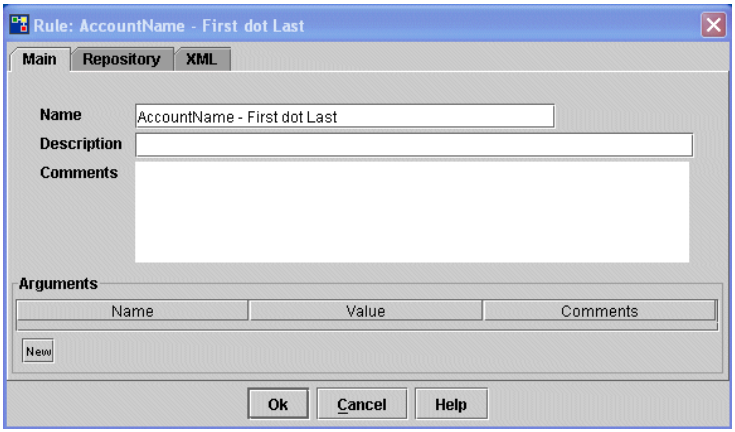
Select an item in the tree or diagram view, and then right-click to display the action menu selections that are available for that item.

## 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-24** Rule Dialog (Main Tab View)




You use the options on this dialog to define a rule, as follows:

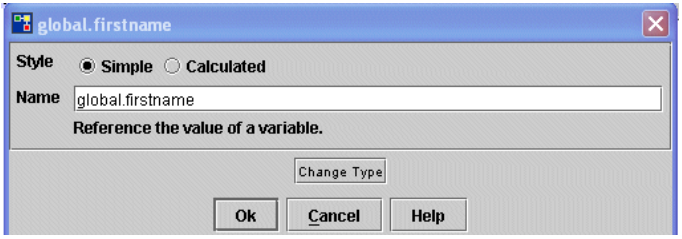
- **Name**: Automatically displays the selected rule name, which is the name displayed in the Identity Manager interface.
- **Description** (optional): Specify text describing the purpose of the rule.
- **Comments**: Specify text inserted into the rule body with a `<Comment>` element.
- **Arguments**: 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-25** 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*

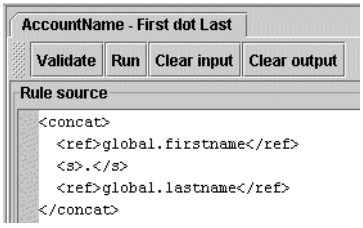
To change the way information displays in diagram view

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.

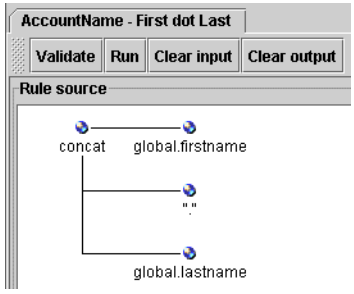
**Figure A-26** XML Display



- **Graphical** – Displays a tree of expression nodes. This display type provides a structural overview.

**NOTE** To conserve space, Figure A-27 shows only part of the selected rule.

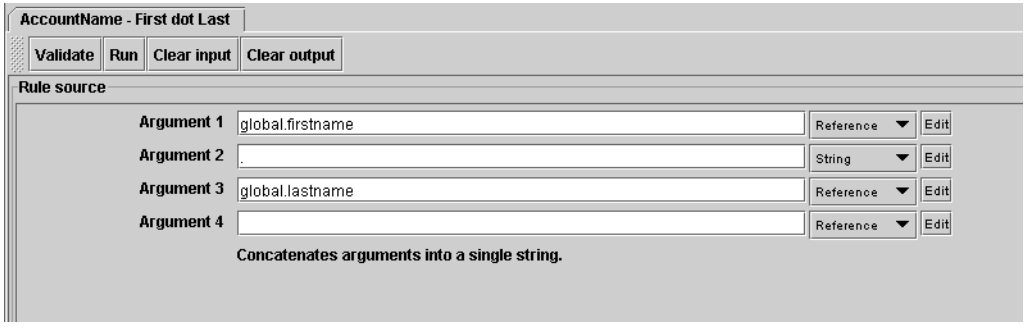
**Figure A-27** Graphical Display




- **Property Sheet** – Displays a list of properties, some of which can be edited directly.

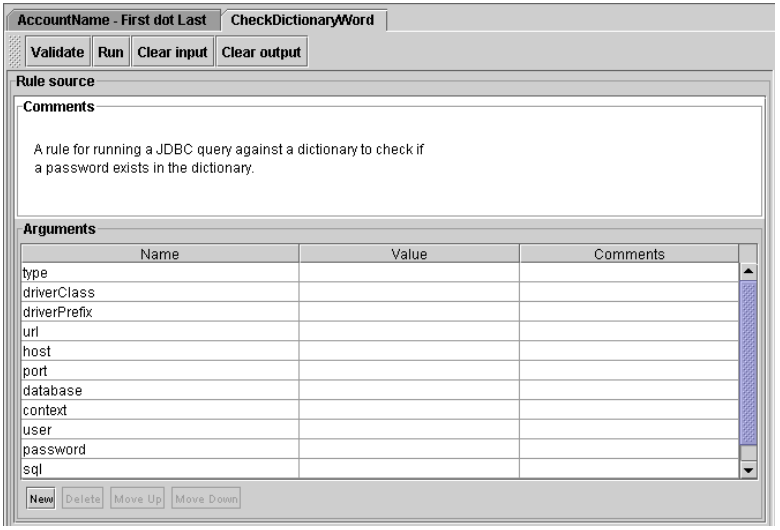
  You may be required to launch another dialog for other properties. For efficiency, use the Property Sheet display type when creating new expressions. (You can enter expression arguments more rapidly in this view compared to using the graphical view.)

**Figure A-28** Property Sheet Display



- **Configuration** – Displays argument information listed in a property-sheet style. (See Figure A-29.) Also lists any comments that the rule creator used to describe the rule in the database.

**Figure A-29** Configuration Display



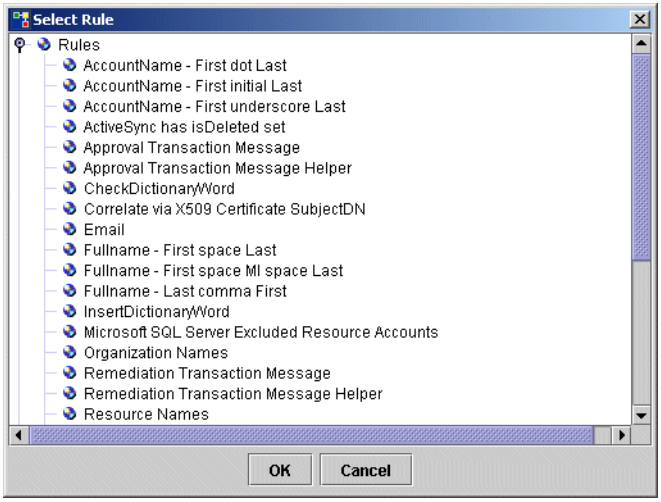

## Browsing Rules

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-30), expand the Rules node to display the available rules.

**Figure A-30** 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:

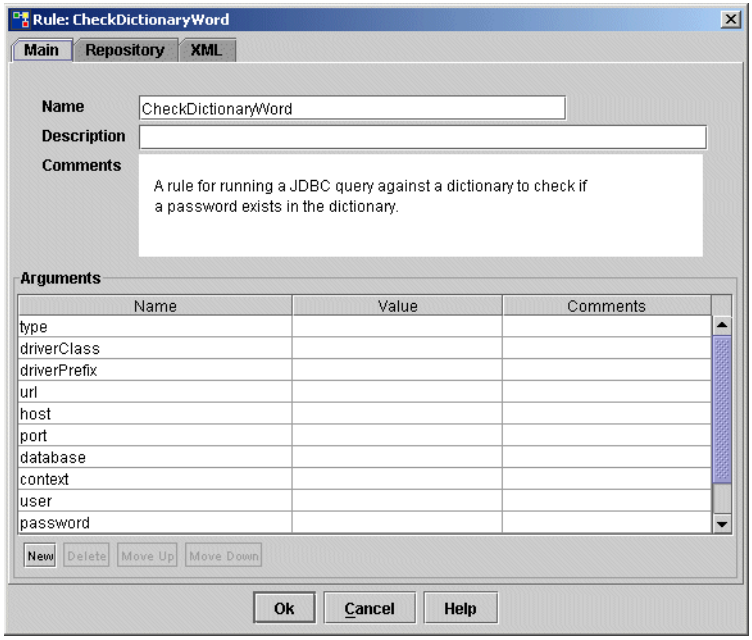
- Main
- Repository
- XML

### *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-31).

**Figure A-31** 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-32** Repository Tab Display

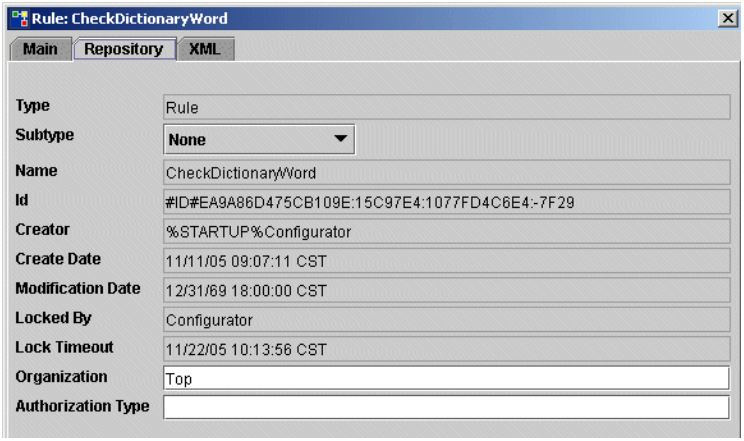



**Table A-2** Fields on the Repository Tab

| Field | Description |
|---|---|
| Type | Identifies the type of repository object. This value is Rule. |
| Subtype | Identifies a subtype, if relevant. Rule subtypes are currently implemented within the Reconciliation interface only.<br>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.

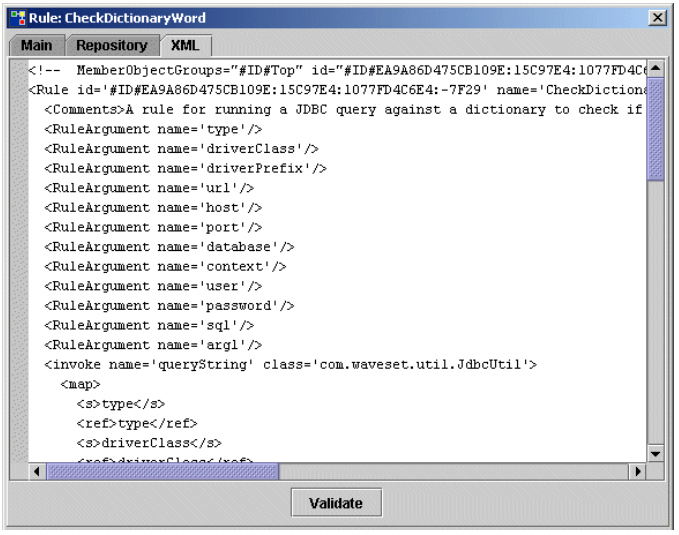
- **Organization**: Enter a new organization assignment into the text field.
- **Authorization Type**: Enter a new authorization type into the text field.

> **NOTE** For an example rule subtype, see "ExcludedAccountsRule" on page 48.

### *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-33** XML Tab Display

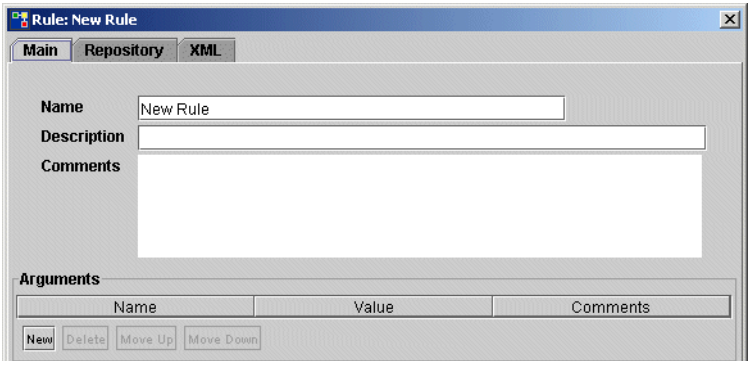


# Creating a New Rule

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-34** 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.

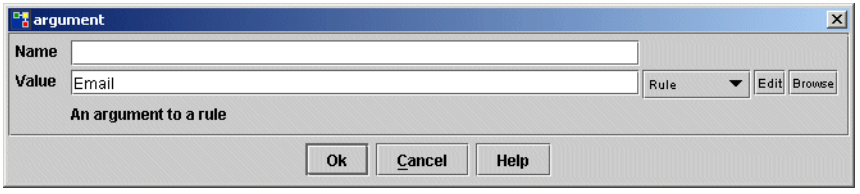
Additional information about these dialogs is provided in the following sections.

**NOTE** For more information about rule structure, see "Understanding Rule Syntax" on page 13.

### *Argument Dialogs*

You use an Argument dialog to access and define rule arguments.

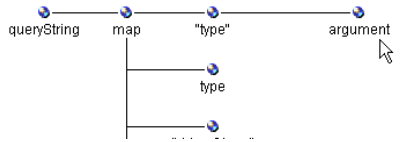
**Figure A-35** Argument Dialog




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-36** Double-Click an Argument Node



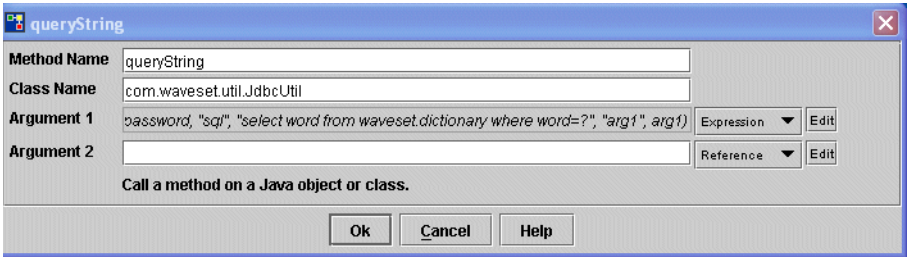

The Argument dialog provides the following basic options:

- **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-37** 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-38** Select Type Dialog


Valid argument types are listed in the following table.

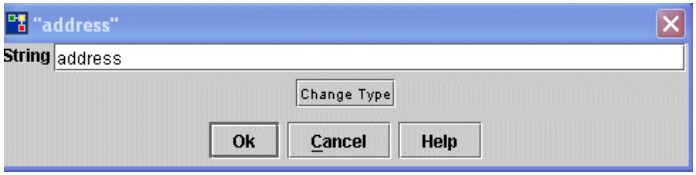
**Table A-3** Valid Argument Types

| 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-39** 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<br>• Create a reference<br>• Assign a value to a variable or an attribute of an object |

**Table A-4** Element Types Representing XPRESS Function Categories *(Continued)*

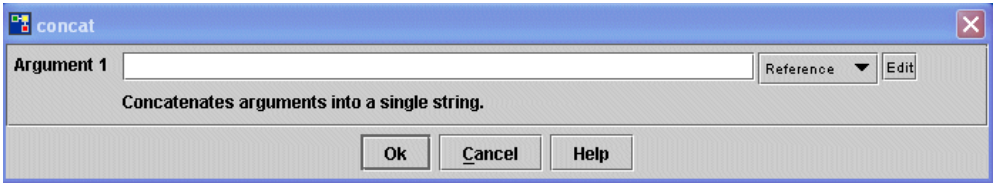
| Menu Options | XPRESS Functions/Additional Actions You Can Invoke... |
|---|---|
| Math | `add`, `sub`, `mult`, `div`, `mod` |
| Control | `switch`, `case`, `break`, `while`, `dolist`, `block` |
| Rule | • Create new rule<br>• Create argument |
| Other<br>(functions, object access, diagnostics) | Displays further options:<br>• Functions includes define function, define argument, and call function<br>• Object access includes the `new`, `invoke`, `getobject`, `get`, and `set` functions.<br>• 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-40** 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.

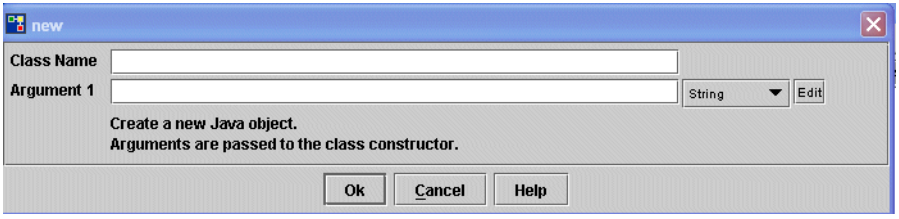
Where *option* can be any of the following options described in Table A-5:

**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.<br>The first argument must be a List, GenericObject, or Object, and the second argument must be a String or Integer.<br>• If the first argument is a List, the second argument is coerced to an integer and used as a list index.<br>• If the first argument is a GenericObject, the second argument is coerced to a String and then used as a path expression.<br>• 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-41** new Dialog

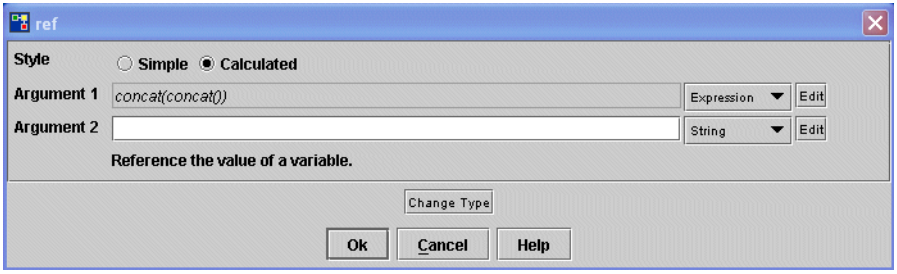


### *Editing Element Details*

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-42 shows the variable window for a `ref` statement.

**Figure A-42** 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.<br><br>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

**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

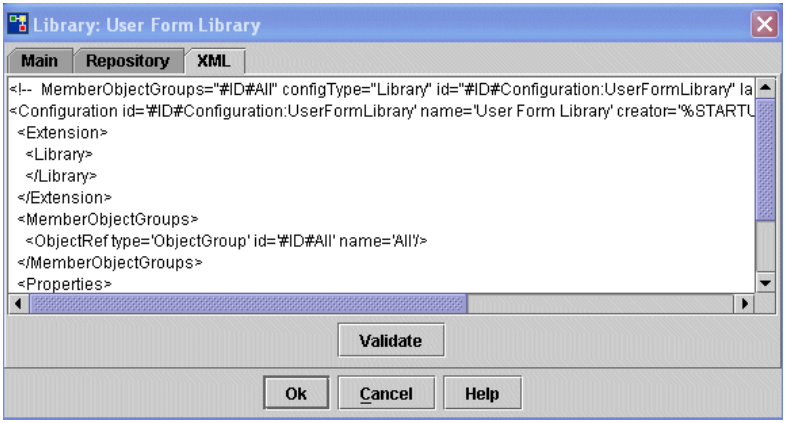
Perform the following steps to select a rule library to view or edit:

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-43** 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**

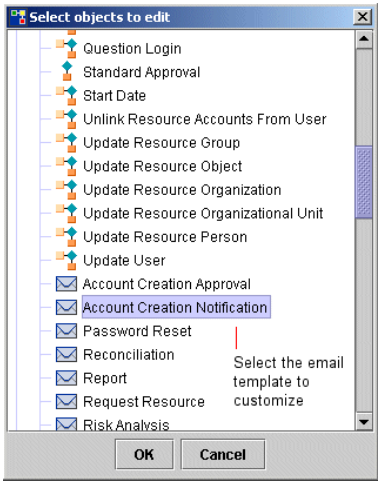
- Illustrations provided in this example may differ slightly from those you see when you load a process. Discrepancies can result in different component positioning or the selective removal of process actions that are unimportant to the example.
- Though you can perform many tasks from the tree view 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-44), select the Account Creation Notification template and then click OK.

**Figure A-44** 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.

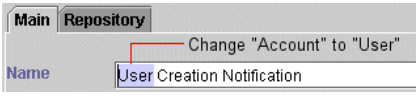
   A copy of the email template appears in the list view.

| **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.

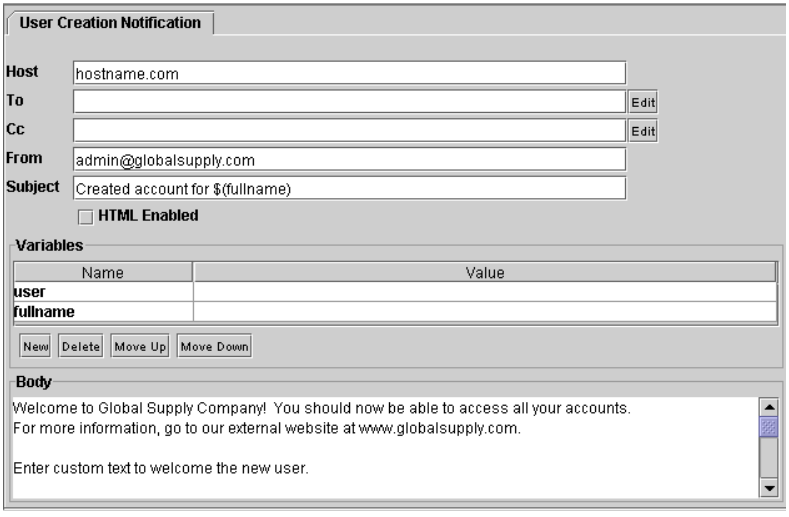
6. Change the template name by typing **`User Creation Notification`** in the Name field.

**Figure A-45** Renaming the New Template




7. In the newly created User Creation Notification template, modify the Subject and Body fields as needed.

**Figure A-46** 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

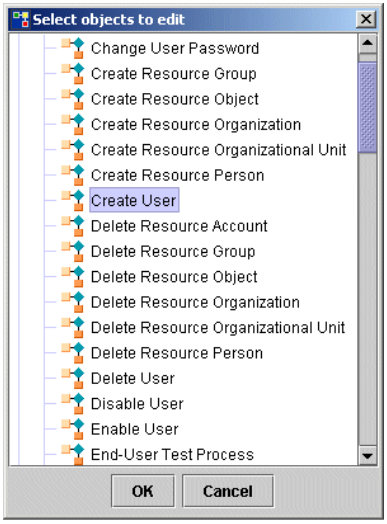
Use the following steps to modify the Create User workflow process to use the new email template:

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**.

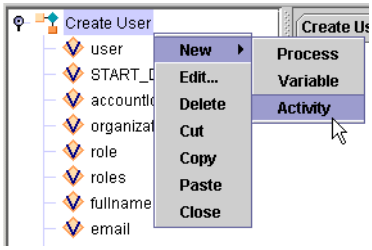
**Figure A-47** Loading the Workflow Process




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-48** Creating and Naming an Activity




A new activity, named `activity1`, displays at the bottom of the activities list in the tree view.

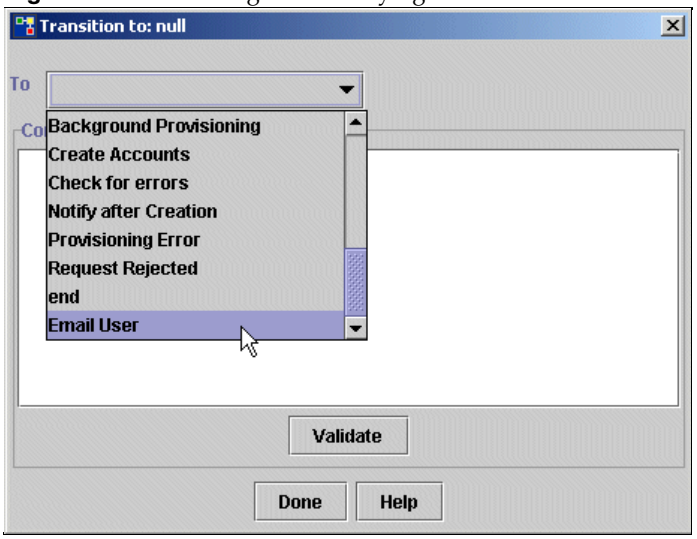
4. Double-click `activity1` to open the activity dialog.
5. Type **`Email User`** in the Name field to change the activity name.

   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.

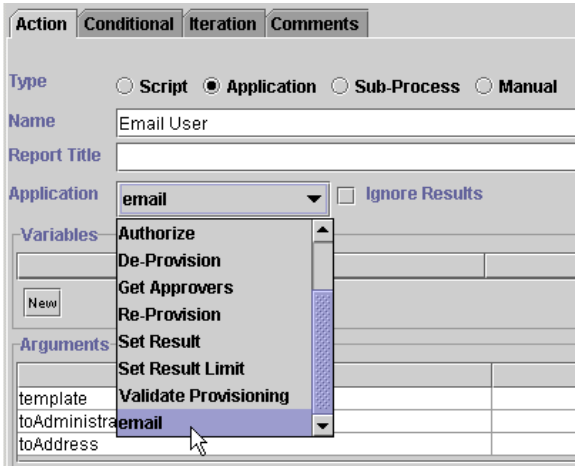
**Figure A-49** Creating and Modifying Transitions




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.

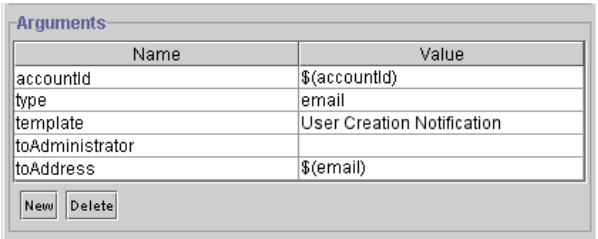
15. Select **email** from the Application menu.

**Figure A-50** 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-51** 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
- Using the Debugger Main Window
- Stepping through an Executing Process
- Getting Started
- Debugging Workflows
- Debugging Forms

# 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 310 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

You must disable the `serverSettings.server-a.debugger.enabled` property in production to prevent someone from accidently connecting a debugger to the application server.

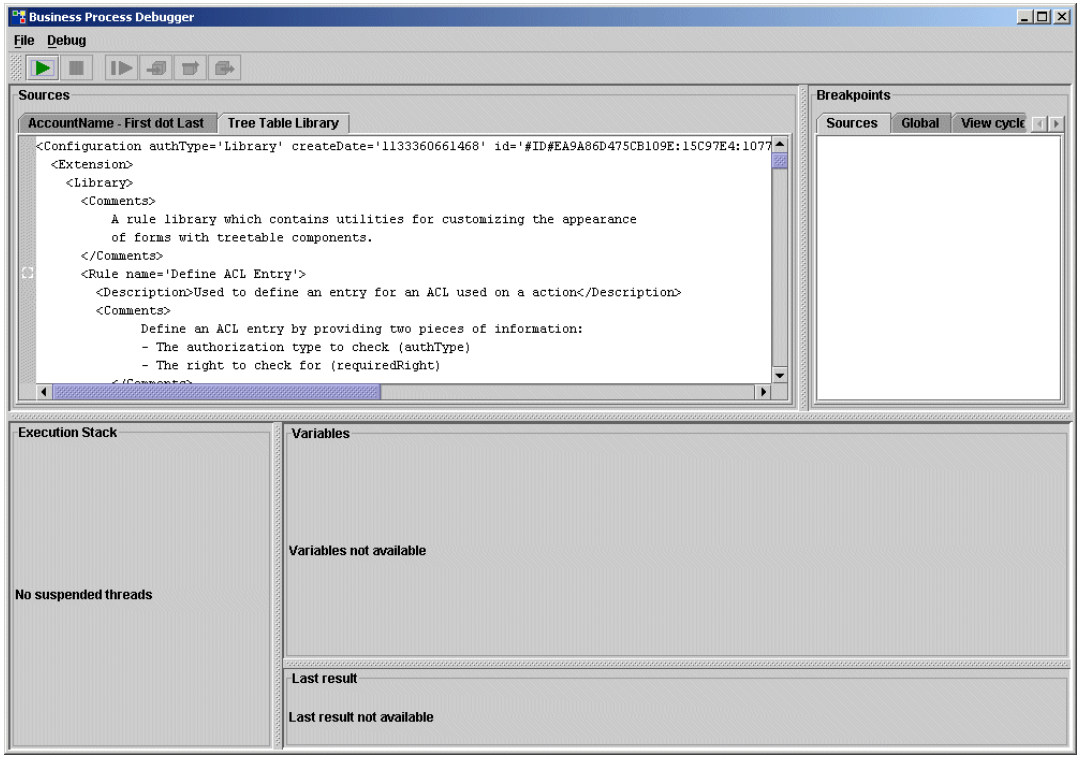
To disable the debugger, set the System Configuration object property `serverSettings.<server>.debugger`. to `enabled=false`.

## 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-52** BPE Debugger: Main Window




**NOTE** The BPE debugger provides numerous keyboard shortcuts for performing tasks. See "Using Keyboard Shortcuts" on page 260 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.

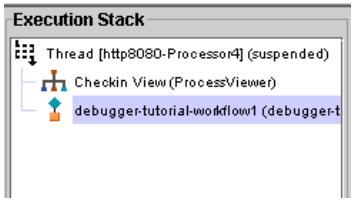
**Figure A-53** BPE Debugger Main Window Source Panel


## 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.

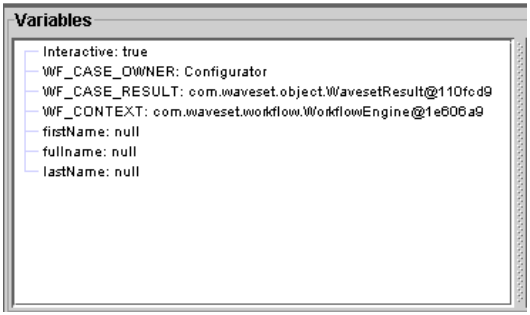
**Figure A-54** BPE Debugger Main Window Execution Stack Panel



## 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-55** 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-56** 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)

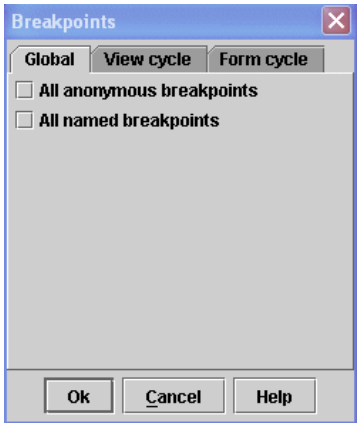
Access these breakpoint types by selecting the designated tab.

- 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 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.

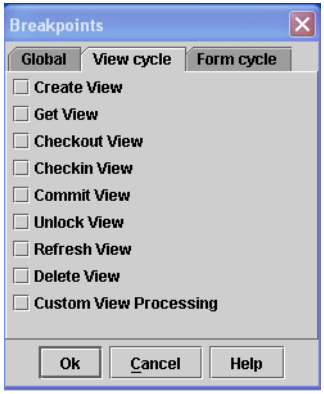
(Optional) Select a global setting, and click OK.

**Figure A-57** BPE Debugger Breakpoints Panel: Global Tab



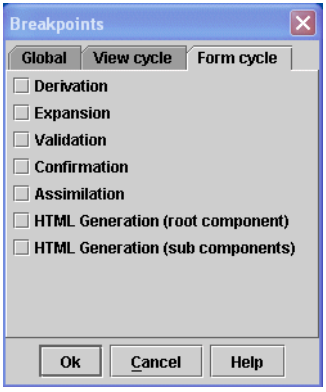

- Select the View cycle tab to set code breakpoints based on the view processing that occurs during process execution. The most commonly invoked view operations are listed in this dialog. Each of the listed view operations are available on each view.

**Figure A-58** 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-59** 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* must be 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*

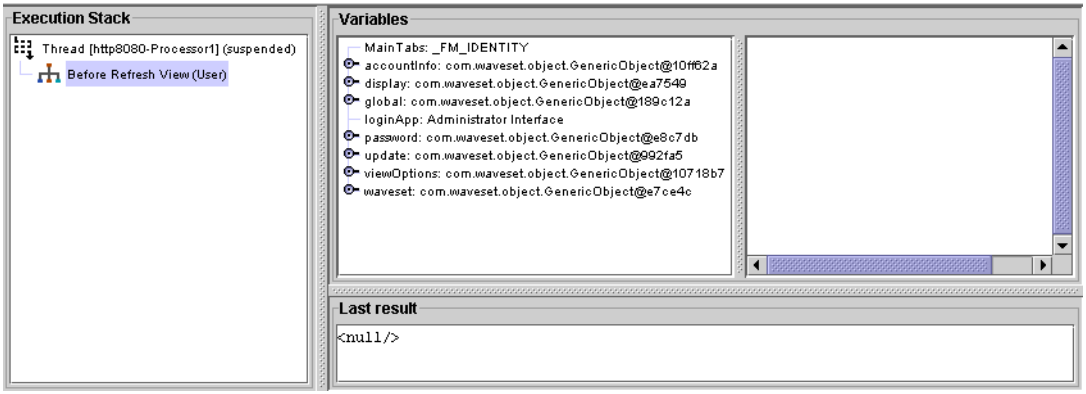
To create a new user:

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-60** 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`.

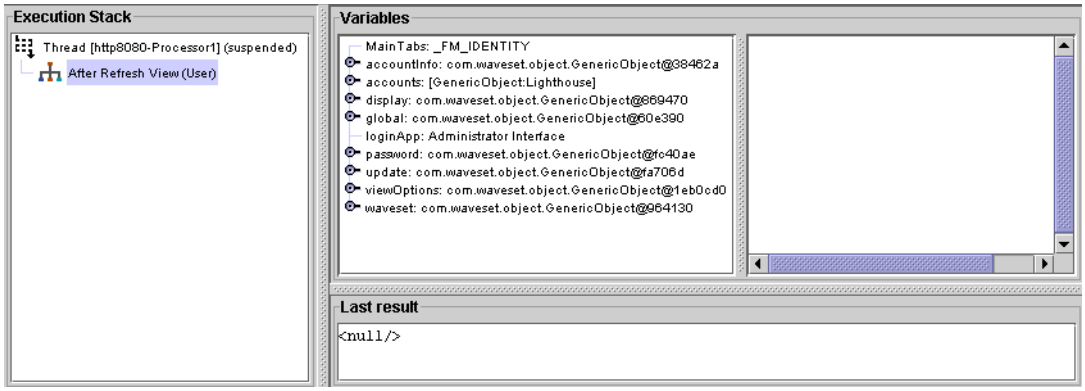
### *Viewing After Refresh View Results*

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-61** 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.

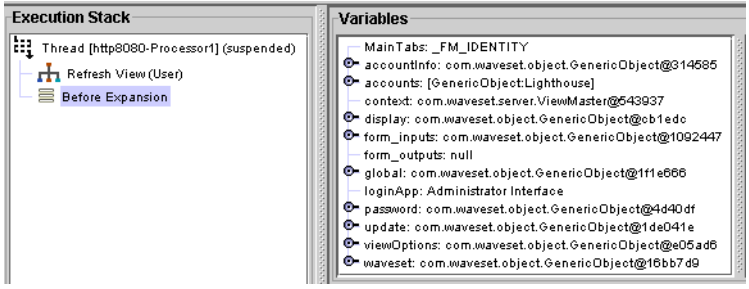
### *Stepping Through the Form*

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-62** Example 1: Debugging Suspended Before First Expansion Pass



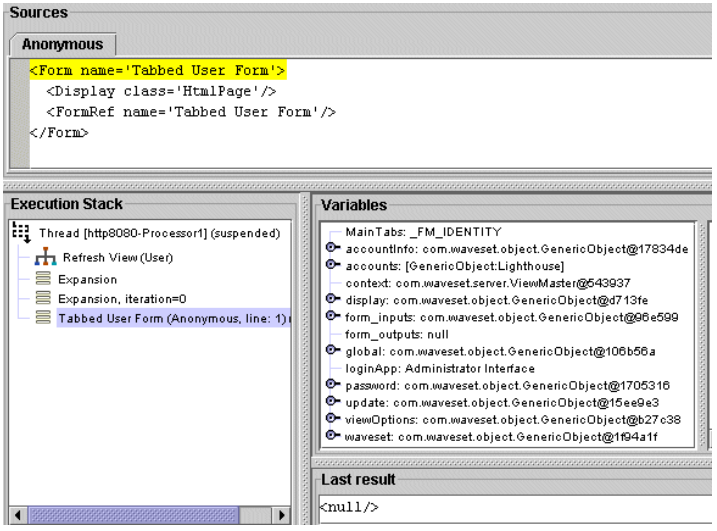
2. Click Step-Into again.

   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-63** 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.

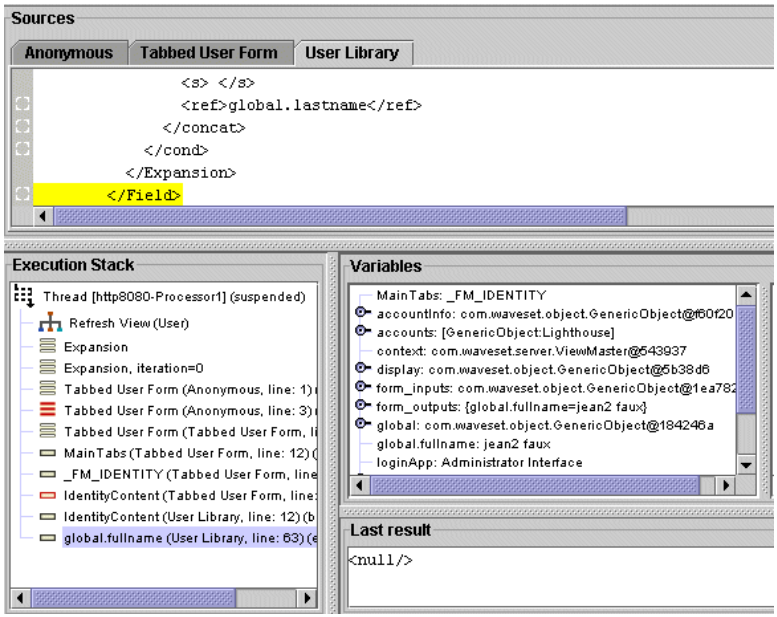
5. Continue to click Step-Into until you reach `<Field name='global.fullName'>` (Approximately 20 to 30 step-into operations.)

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-64** 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.<br><br>This is a logical state, which does not mean that the Java thread is currently executing it. The action currently being executed is 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 shows 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.

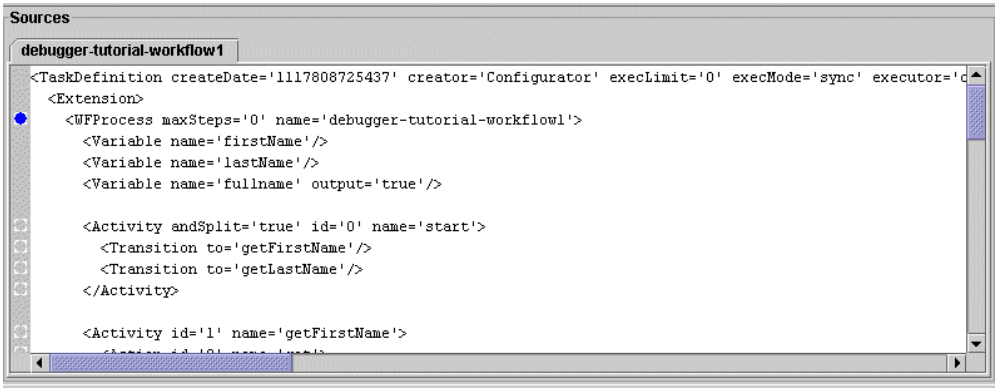
### *Step One: Launch the Process*

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-65** Setting the First Breakpoint

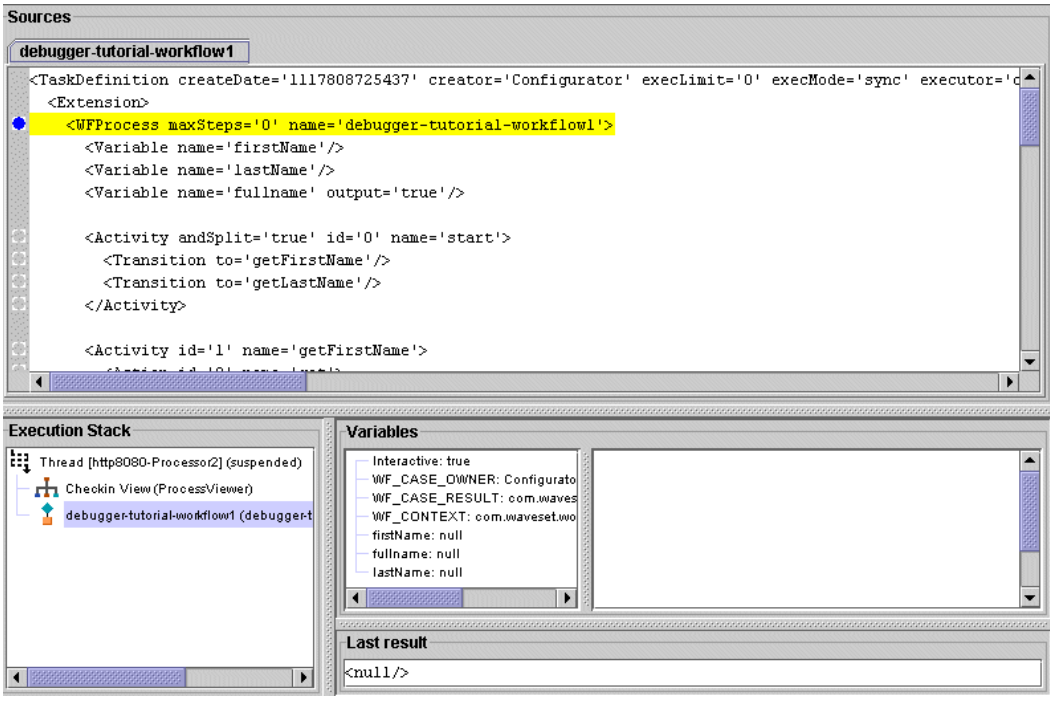



3. Click the box adjacent to the `<WFProcess>` tag to set a breakpoint at the start of the workflow.
4. Log in to Identity Manager and select Tasks > Run Tasks.

5. Click debugger-tutorial-workflow1.

   The debugger frame indicates that debugging has halted at your breakpoint.

**Figure A-66** 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.

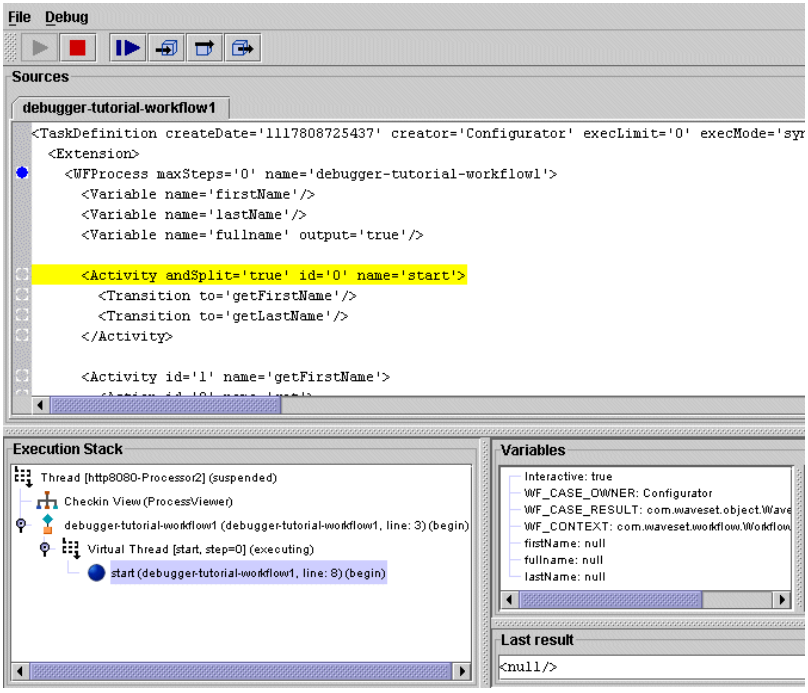
### *Step Two: Start Execution*

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-67** 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:

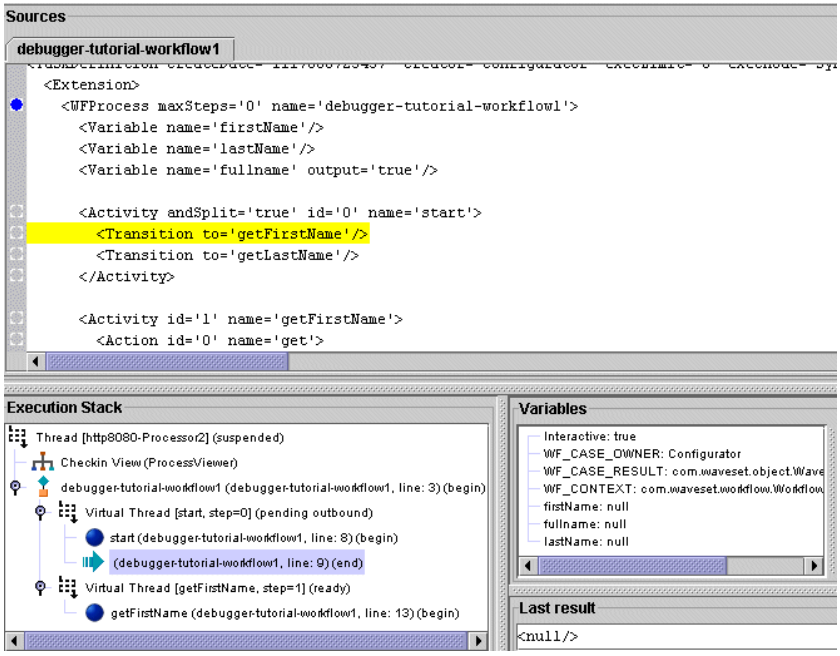
1. Click Step-Into.

   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-68** 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`.

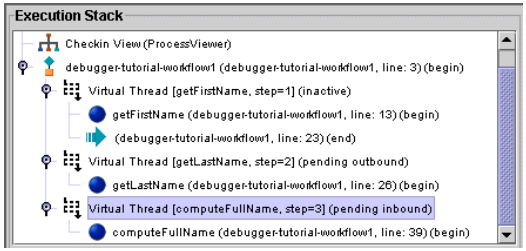
7. Click Step-Into.

   `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-69** 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.

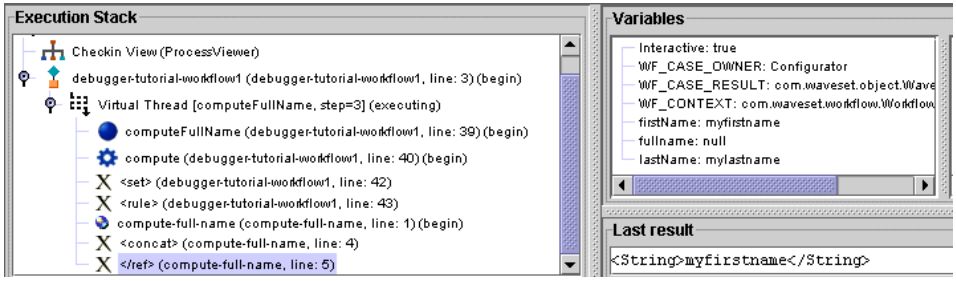
2. Click Step-Into.

   `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-70** 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.

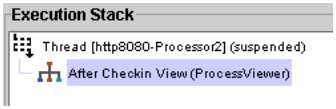
7. Click Step-Into four more times. `end` goes to ready, then executing.

   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-71** 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 shows how to debug a sample workflow containing a manual action and a form.

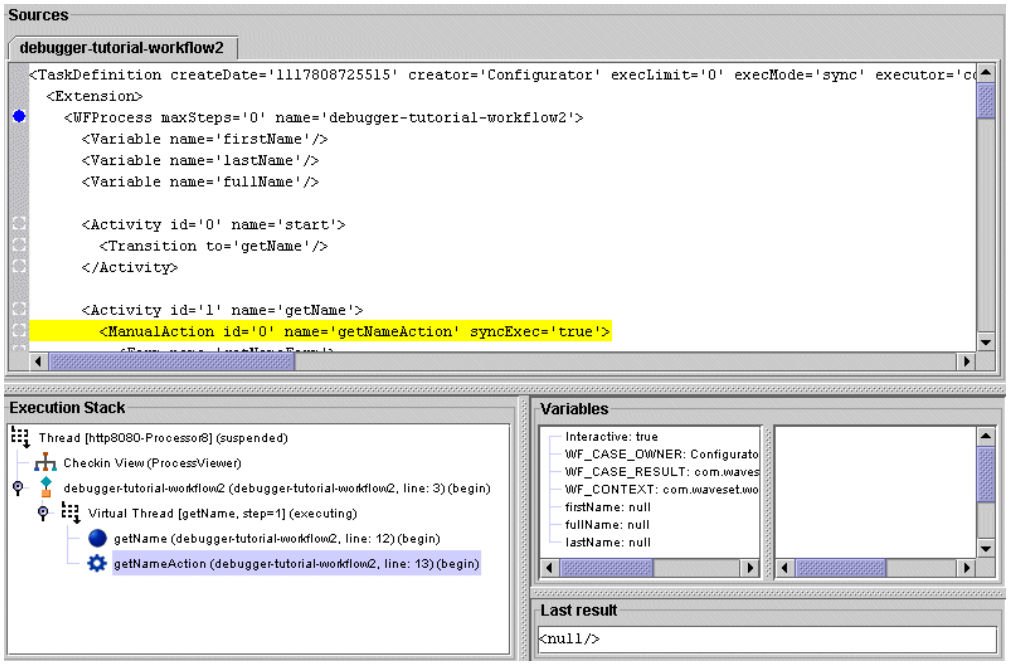
Use `workflow2` from the debugger tutorial files and perform the following steps:

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'>).

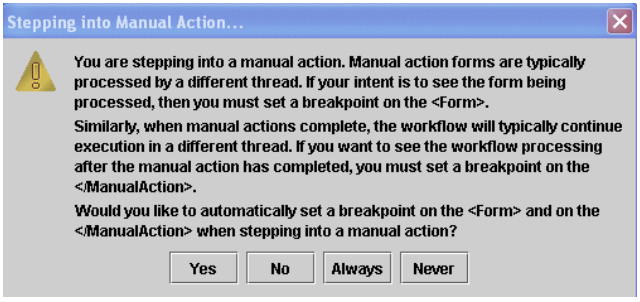
**Figure A-72** Stepping Into a Manual Action




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.

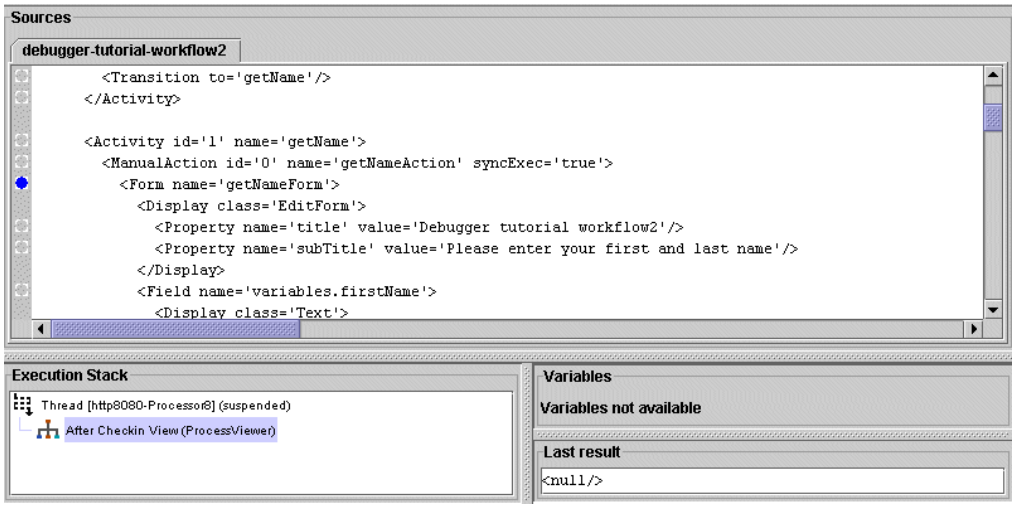
**Figure A-73** Stepping Into Manual Action Dialog



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.

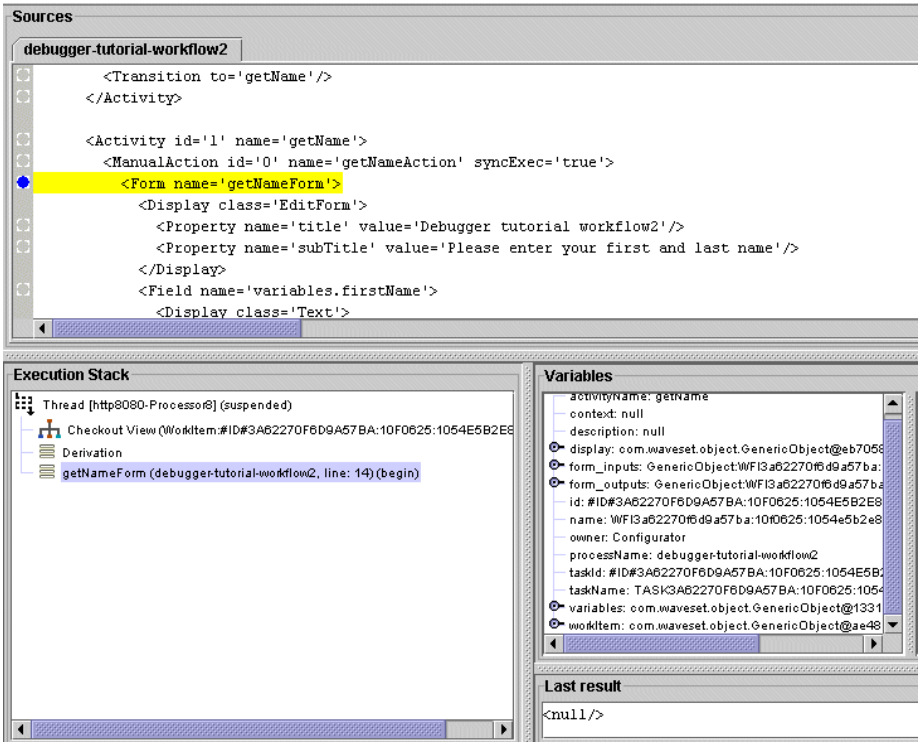
**Figure A-74** Breakpoint Marking Start of Form




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-75** 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.

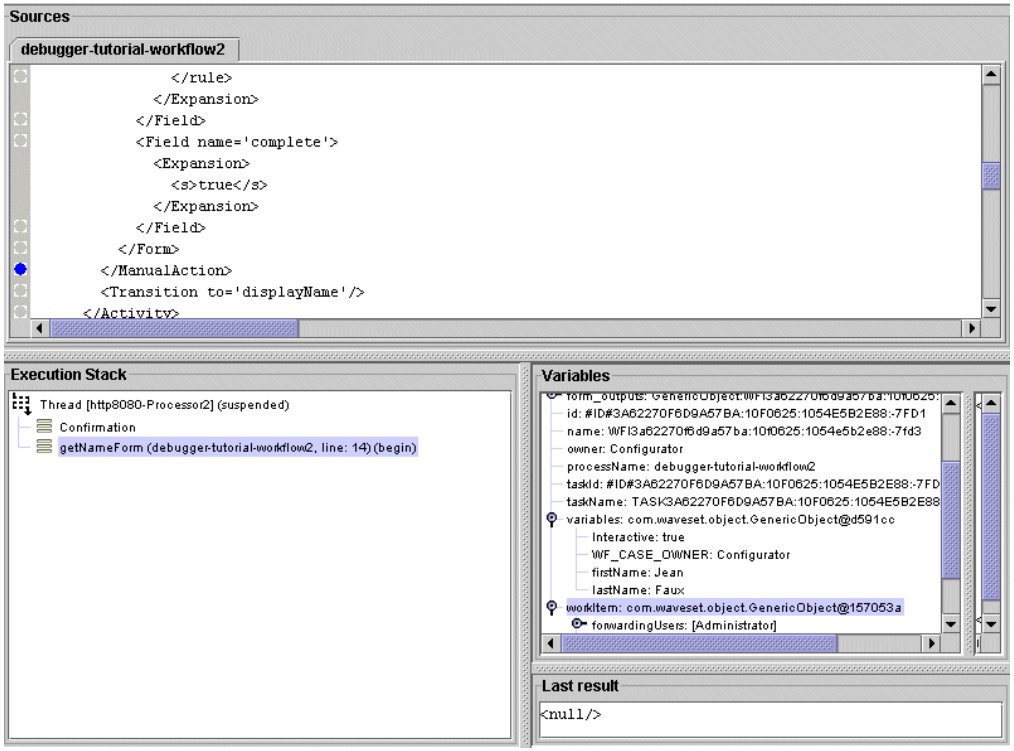
3. Return to your browser window, and enter your first and last name as prompted, and click Save.

   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-76** 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.

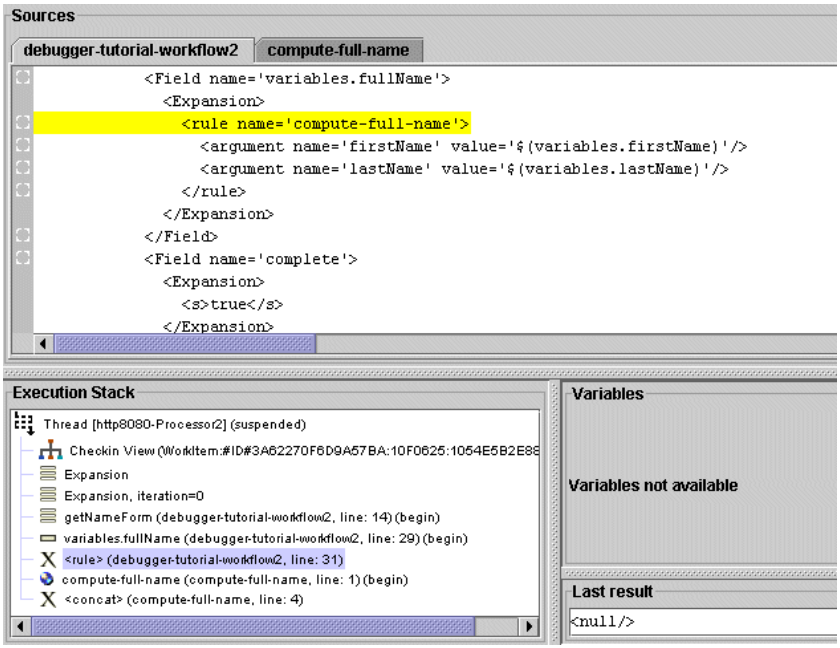
1. Click Continue to skip the Validation phase.

   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-77** Stepping Into Rule Processing



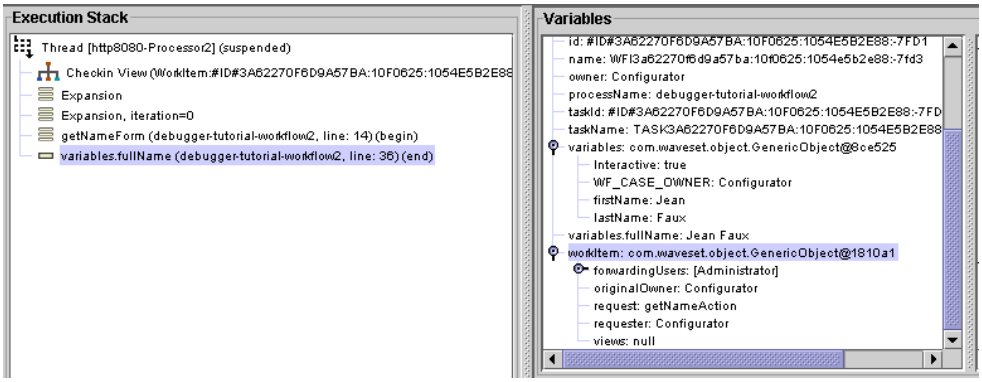

3. Click Step-Into five times, and the debugger has now stepped into the `<rule>` element.
4. Click Step-Into seven times or until the debugger reaches the `</Rule>` element.

   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-78** Debugger Displaying Completed Execution of variable.fullName

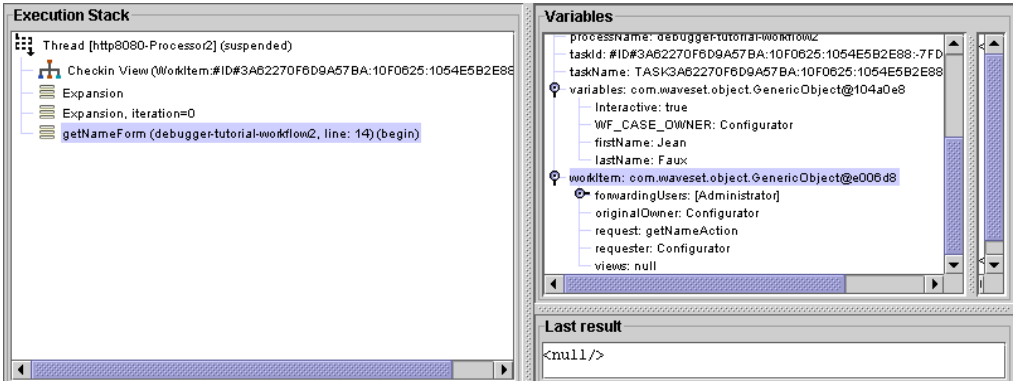



In general, `form_inputs` identifies the view, and `form_outputs` contains data to be assimilated back into the view. However, not all forms are necessarily tied to a view (for example, Active Sync forms). The form engine is a general data mapping engine, mapping from form inputs to form outputs. The 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-79** 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 has the pass name.

## 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



# L

# M



# N



# O



# P

# R

# S

# T



# U



# V



# W



# X