Oracle® Waveset 8.1.1 Deployment Guide

Part No: 821–0093
July 2010

100730@24661

Contents

Preface 11

1 **Working with Attributes** 17

Related Chapters 17

What are Attributes? 17

Types of Attributes 17

Using Attribute Conditions 22

Attribute Condition Operators 23

Implicitly `AND`ed 24

Using Secret Attributes 25

2 **Working with Authorization Types** 27

What are Authorization Types? 27

How Waveset Uses Authorization Types 28

Why Use Authorization Types? 28

Using Authorization Types 29

Configuration:AuthorizationTypes Object 29

AuthType Element 29

Authorization Types and Capabilities 30

AdminGroups Objects 30

EndUser Capability 30

Creating an Authorization Type 31

Assigning an Authorization Type to a Repository 31

Example: Setting End-User Authorization Types 31

Example: Using Authorization Types to Restrict Visibility on Resources 32

Example: Granting Access to a Specific Part of Waveset 33

3 Data Loading and Synchronization 35
Types of Data Loading 36
Discovery 36
Reconciliation 38
Active Sync 40
Summary of Data Loading Types 40
Load Operation Context 41
Managing Reconciliation 42
Reconciliation Policy 42
Resource Scheduling 48
Reconcile Configuration Object 48
Managing Active Sync 49
How Active Sync-Enabled Adapters Work 49
Using Forms 53

4 Dataloading Scenario 61
Assessing Your Environment 61
Choosing the First Resource 62
Choosing the First Data Loading Process 64
Using Load from File 64
Using Load from Resource 66
Using Create Bulk Actions 66
Using Reconciliation 67
Preparing for Data Loading 68
Configuring an Adapter 68
Setting Account ID and Password Policies 68
Creating a Data Loading Account 69
Assigning User Forms 69
Linking to Accounts on Other Resources 71
Defining Custom Correlation Keys 72
Creating Custom Rules 73
Manually Linking Accounts 74
Example Scenarios 76
Active Directory, SecurID, and Solaris 76
LDAP, PeopleSoft, and Remedy 80

Expedited Bulk Add Scenario 85

5 **Data Exporter** 87
What is Data Exporter? 87
Exportable Data Types 88
Data Exporter Architecture 89
Planning for Data Exporter 91
Database Considerations 91
Export Server Considerations 93
Loading the Default DDL 94
DB2 94
MySQL 94
Oracle 95
SQL Server 95
Upgrading Data Exporter 96
Customizing Data Exporter 97
Waveset ObjectClass Schema 97
Export Schema 98
Modifying the Warehouse Interface Code 99
Generating a New Factory Class 100
Adding Localization Support for the WIC 101
Troubleshooting Data Exporter 101
Beans and Other Tools 101
Model Serialization Limits 101
Repository Polling Configuration 102
Tracing and Logging 102

6 **Integrating Waveset and SAP BusinessObjects Access Control** 103
External Policy Check Integration 104
Supported Web Services 106
Implementing a External Policy Check Integration 107
Managing Compliance Violations 107
External Resource Integration 108
Supported Web Services 110
Implementing a New External Resource Integration 110

Migrating from Access Enforcer Version 5.1 or 5.2 111
SPML 1.0 Integration 112
SPML Configuration 112
SAP Access Control Configuration Information 113
SPML Request Behavior 114
User Create Password Notification 115
Audit Log Support 115

7 Configuring User Actions 117
Adding Custom Tasks 117
Setting Up Custom Task Authorization 117
Adding a Task to the Repository 119
Configuring User Actions 121
▼ To Configure User Actions 122

8 Private Labeling of Waveset 127
Private Labeling Tasks 127
Architectural Features 127
Style Sheets 128
Default Text 128
Text Attributes 128
Default Style Settings 129
Customized File 129
JSP Files 129
WPMessages_en.properties File 129
Customizing Headers 129
Changing Header Appearance 129
Customizing Waveset Pages 130
Creating a Custom Message Catalog 130
Customizing the Home Page 130
Changing Default Information Displayed in the Waveset User Interface Home Page 133
Changing the Appearance of the User Interface Navigation Menus 134
Changing Font Characteristics 134
Sample Labeling Exercises 135
Replacing the Waveset Logo with a Custom Logo 135

Changing Masthead Appearance 136
Changing Navigation Tabs 136
Changing Tab Panel Tabs 137
Changing Sorting Table Header 139
Changing User / Resource Table Component 139
Changing Waveset Behavior on Commonly Used Pages 140

9 Customizing Message Catalogs 143
Advantages of Custom Message Catalogs 143
How Waveset Retrieves Message Catalog Entries 143
Message Catalog Format 143
Creating a Customized Message Catalog 144
Example Message Catalog 145

10 Developing Custom Adapters 147
Before You Begin 147
Intended Audience 148
Important Notes 148
Related Documentation 148
What is a Resource Adapter? 149
What Are Standard Resource Adapters? 149
What Are Active Sync-Enabled Resource Adapters? 150
What is a Resource Object? 155
What is a Resource Adapter Class? 155
Preparing for Adapter Development 156
Become Familiar with Adapter Source Code 156
Profile the Resource 170
Decide Which Classes and Methods to Include 173
Review the REF Kit 173
Set Up the Build Environment 174
Writing Custom Adapters 176
Process Overview 176
Rename the Skeleton File 178
Edit the Source File 178
Map the Attributes 179

Specify the Identity Template 180
Write the Adapter Methods 180
Configure the Adapter to Support Pass-Through Authentication 192
Define the Resource Object Components 193
Installing Custom Adapters 199
▼ To Install a Customized Resource Adapter 199
Testing Custom Adapters 200
Unit Testing Your Adapter 200
Compatibility Testing Your Adapter 201
Testing the Resource Object 217
Troubleshooting Custom Adapters 219
Maintaining Custom Adapters 219

11 Optimistic Checkouts 221
Principles of Optimistic Checkouts 222
Detecting Conflicts 223
Implementing Optimistic Checkouts 226

12 Editing Configuration Objects 231
Data Storage 231
Object Naming Conventions 232
Viewing and Editing Configuration Objects 233
IDM Schema Configuration Object 234
UserUIConfig Object 237
RepositoryConfiguration Object 237
WorkItemTypes Configuration Object 238
SystemConfiguration Object 239
Role Configuration Object 242
End User Tasks Object 245
Refreshing User Objects 245

13 Enabling Internationalization 247
Architectural Overview 247
Typical Entry 248

Enabling Support for Multiple Languages 249
Step One: Download and Install Localized Files 249
Step Two: Edit the `Waveset.properties` File 250
Maintaining ASCII Account IDs and Email Addresses During Anonymous Enrollment Processing 251
▼ To Maintain Account ID and Email Address Values 251

Index 253

Preface

This *Oracle Waveset 8.1.1 Technical Deployment Guide* publication provides an overview of the reference and procedural information you will use to customize Oracle Waveset for your environment.

Oracle Waveset was formerly known as Sun Identity Manager. This release of Oracle Waveset follows Oracle's acquisition of Sun Microsystems. Some parts of the product and documentation may still refer to the original Sun Identity Manager product name.

Who Should Use This Book

The *Oracle Waveset 8.1.1 Technical Deployment Guide* was designed for deployers and administrators who will create and update workflows, views, rules, system configurations and other configuration files necessary to customize Waveset 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.

How This Book Is Organized

This book is organized into these chapters:

- Chapter 1, “Working with Attributes,” introduces Identity attributes and how to use this feature to streamline the data flow through your Waveset deployment.
- Chapter 2, “Working with Authorization Types,” describes how to assign authorization rights to Waveset objects.
- Chapter 3, “Data Loading and Synchronization,” presents an overview of the reconciliation and other mechanisms for loading account information into Waveset. Reconciliation compares the set of users defined in Waveset to the set of accounts that are defined on an Waveset resource.
- Chapter 4, “Dataloading Scenario,” provides tips to consider when preparing to load account information into IdentityManager, including sample scenarios that illustrate some of the issues that you might encounter.

- Chapter 5, "Data Exporter," describes how to plan for and implement the Data Exporter feature.
- Chapter 6, "Integrating Waveset and SAP BusinessObjects Access Control"describes how to implement SAP Access Control.
- Chapter 7, "Configuring User Actions," describes how to add custom tasks to the Waveset Administrator Interface and configure user actions that you can execute from two areas of the interface.
- Chapter 8, "Private Labeling of Waveset," describes how to customize colors, logos, and header and footer content to meet the style standards of your organization.
- Chapter 9, "Customizing Message Catalogs," describes how to create a message catalog so that you can override the default text displayed on the user and administrator interfaces.
- Chapter 10, "Developing Custom Adapters," describes how to create custom Waveset resource adapters that are tailored to your company or customers.
- Chapter 11, "Optimistic Checkouts," describes how to allow multiple processes to access a User object simultaneously.
- Chapter 12, "Editing Configuration Objects," provides an overview of configuration objects and a discussion of the `UserUIConfig` object.
- Chapter 13, "Enabling Internationalization," provides information on configuring Waveset to use multiple languages or display a language other than English.

Related Books

Oracle provides additional documentation and information to help you install, use, and configure Waveset. The Oracle Waveset 8.1.1 library includes the following publications:

Primary Audience	Title	Description
All Audiences	*Oracle Waveset 8.1.1 Overview*	Provides an overview of Waveset features and functionality. Provides product architecture information and describes how Waveset integrates with other Sun products, such as Sun Open SSO Enterprise and Role Manager.
	Oracle Waveset 8.1.1 Release Notes	Describes known issues, fixed issues, and late-breaking information not already provided in the Waveset documentation set.

Primary Audience	Title	Description
System Administrators	*Oracle Waveset Installation*	Describes how to install Waveset and optional components such as the Oracle Waveset Gateway and PasswordSync.
	Oracle Waveset 8.1.1 Upgrade	Provides instructions on how to upgrade from an older version of Waveset to a newer version.
	Oracle Waveset 8.1.1 System Administrator's Guide	Contains information and instructions to help system administrators manage, tune, and troubleshoot their Waveset installation.
Business Administrators	*Oracle Waveset 8.1.1 Business Administrator's Guide*	Describes how to use Waveset's provisioning and auditing features. Contains information on the user interfaces, user and account management, reporting, and more.
System Integrators	*Oracle Waveset 8.1.1 Deployment Guide*	Describes how to deploy Waveset in complex IT environments. Topics covered include working with identity attributes, data loading and synchronization, configuring user actions, applying custom branding, and so on.
	Oracle Waveset 8.1.1 Deployment Reference	Contains information on workflows, forms, views, and rules, as well as the XPRESS language.
	Oracle Waveset 8.1.1 Resources Reference	Provides information about installing, configuring, and using resource adapters.
	Oracle Waveset Service Provider 8.1.1 Deployment	Describes how to deploy Oracle Waveset Service Provider, and how views, forms, and resources differ from the standard Wavesetproduct.
	Oracle Waveset 8.1.1 Web Services	Describes how to configure SPML support, which SPML features are supported (and why), and how to extend support in the field.

In addition, the `http://docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the archive or search for a specific book title or subject.

Documentation Updates

Corrections and updates to this and other Waveset publications are posted to the Waveset Documentation Updates website:

`http://blogs.sun.com/idmdocupdates/`

An RSS feed reader can be used to periodically check the website and notify you when updates are available. To subscribe, download a feed reader and click a link under Feeds on the right side of the page. Starting with version 8.0, separate feeds are available for each major release.

Related Third-Party Web Site References

Third-party URLs are referenced in this document and provide additional, related information.

Note – 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.

Documentation, Support, and Training

See the following web sites for additional resources:

- Documentation (`http://docs.sun.com`)
- Support (`http://www.oracle.com/us/support/systems/index.html`)
- Training (`http://education.oracle.com`) – Click the Sun link in the left navigation bar.

Oracle Welcomes Your Comments

Oracle welcomes your comments and suggestions on the quality and usefulness of its documentation. If you find any errors or have any other suggestions for improvement, go to `http://docs.sun.com` and click Feedback. Indicate the title and part number of the documentation along with the chapter, section, and page number, if available. Please let us know if you want a reply.

Oracle Technology Network (`http://www.oracle.com/technetwork/index.html`) offers a range of resources related to Oracle software:

- Discuss technical problems and solutions on the Discussion Forums (`http://forums.oracle.com`).
- Get hands-on step-by-step tutorials with Oracle By Example (`http://www.oracle.com/technology/obe/start/index.html`).
- Download Sample Code (`http://www.oracle.com/technology/sample_code/index.html`).

Typographic Conventions

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

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`

`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`**
		`Password:`

aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX system prompt and superuser prompt for shells that are included in the Oracle Solaris OS. Note that the default system prompt that is displayed in command examples varies, depending on the Oracle Solaris release.

TABLE P–2 Shell Prompts

Shell	Prompt
Bash shell, Korn shell, and Bourne shell	`$`
Bash shell, Korn shell, and Bourne shell for superuser	`#`
C shell	`machine_name%`
C shell for superuser	`machine_name#`

CHAPTER 1

Working with Attributes

This chapter presents a conceptual overview of attributes as used in Oracle Waveset deployments. The topics in this chapter include

- "Related Chapters" on page 17
- "What are Attributes?" on page 17
- "Using Attribute Conditions" on page 22
- "Using Secret Attributes" on page 25

Related Chapters

Attributes are manipulated as part of many Waveset operations and are discussed throughout the documentation set. The following chapters contain a substantial amount of information related to attributes:

- Chapter 3, "Waveset Views," in *Oracle Waveset 8.1.1 Deployment Reference* provides an extensive discussion of view attributes, registering attributes, and deferred attributes.
- *Oracle Waveset 8.1.1 Resources Reference* provides information about resource attributes.

What are Attributes?

Attributes are name-value pairs that are used to define and manipulate characteristics of Waveset objects as well as external resources. Waveset components such as forms, workflows, and rules call attributes as an essential part of accessing and transforming data in their regular operations.

Types of Attributes

Although objects in an Waveset deployment can contain a variety of attributes, you typically work with the attribute types described in the following sections:

- “Summary Attributes” on page 18
- “Queryable Attributes” on page 18
- “Inline Attributes” on page 18
- “Extended Attributes” on page 19
- “Operational Attributes” on page 20
- “View Attributes” on page 21
- “Resource User Attributes” on page 21
- “Identity System User Attributes” on page 21
- “Other Standard Attributes” on page 22

Summary Attributes

Every persistent object exposes a set of *summary attributes*. Summary attributes are configured in the Waveset Schema Configuration object and contain the values that are returned for each item in the result of a list operation. Each `PersistentObject` subclass can extend the default set of attributes by overriding the `getSummaryAttributes` method.

Summary attributes are typically single-valued, because there is a limit to the total length of summary attributes when serialized to a string.

You configure these attributes for user objects directly through the Waveset Schema Configuration object. For more information, see “IDM Schema Configuration Object” on page 234.

Note – The `SummaryAttrNames` section of `UserUIConfig` is no longer used in Waveset.

Queryable Attributes

Every persistent object exposes a set of *queryable attributes*. Queryable attributes contain the set of values used for filtering and matching, and these attributes are configured in the Waveset Schema Configuration object. Queryable attributes can be multi-valued.

Note – The `QueryableAttrNames` section of `UserUIConfig` is no longer used in Waveset.

Inline Attributes

You can designate up to five queryable attributes for each object type as *inline attributes*. Inline attributes are configured in the Waveset Repository Configuration object.

Note – Inline attributes are no longer configured in `UserUIConfig`.

Designating an attribute as *inline* asks the data store to optimize the performance of queries against that attribute.

Waveset typically stores each value of a queryable attribute as a row in an attribute table that is separate from the main object table. The attribute table can be joined to the object table to select objects that match an `AttributeCondition`.

Waveset stores the value of an *inline* attribute, however, directly in the object table for that type. Designating an attribute as inline allows Waveset to generate more efficient SQL. A column expression on the main object table is faster than a JOIN to (or an EXISTS predicate against) the corresponding attribute table. This improves the performance of any query against the attribute.

You can characterize inline attributes as follows:

- **An inline attribute must be single-valued** because its value is stored in a single column of the parent row in the object table.
- **Up to five queryable attributes can be inline for a type** because the object table contains only five columns that can be used to store arbitrary attribute values.
- **The same set of queryable attributes is designated as inline for every instance of a type** because the correspondence between the column value and the name of an attribute is specified only by the configuration. That is, the configuration of inline attributes for a type is the only way the repository knows which attribute is stored in which column.

Extended Attributes

Extended attributes are just attributes that are not built-in, such as `employeeNumber` for User. Most customers want to be able to query by `employeeNumber`, so you can add this attribute as a queryable extended attribute through the configuration.

Note – It is a *best practice* to prefix extended attributes with a deployment-specific prefix to prevent potential conflicts with new core attributes in future releases of Waveset.

For example, when adding an extended attribute to User to record the `employeeNumber`, use a prefix associated with the company, such as `acme_employeeNumber`. If a future Waveset release incorporates a built-in user attribute named `employeeNumber`, the two attributes will remain distinct. Otherwise the built-in attribute takes precedence.

Because extended attributes are not built-in, these attributes must be in the `<IDMAttributeConfigurations>` section of the `IDM Schema Configuration` object. This section captures the attribute names, syntax (such as `string`, `int`, and `date`), and whether the attribute is single-valued or multi-valued. The `IDMObjectClassConfiguration` captures which attributes are in which object classes because named attributes can actually be in more than one object class, such as `MemberObjectGroups`.

Note – The `IDM Schema Configuration` object is protected with the `IDMSchemaConfig` authType.

Administrators needing to view or edit the Waveset schema for Users or Roles must have the `IDMSchemaConfig AdminGroup` (capability) assigned. The Configurator user has this `AdminGroup` assigned by default.

For more information about `User` extended attributes, see the discussion about the `accounts[lighthouse]` attribute of the User view in the *Views* chapter of Deployment Reference.

You can expose built-in attributes and extended attributes as *queryable* or *summary*. Some built-in attributes have `REFERENCE` syntax, but extended attributes are not allowed to be `REFERENCE`.

The `<Comments>` section of the effective schema contains information about available internal attributes, as well as extended attributes for relevant objectclasses. You can view this information from the Waveset Debug pages by clicking the Display Schema button and selecting ObjectClass Schema from the list.

Note – Extended attributes are supported for User, Role, and extensions of Role only.

Some built-in attribute references for User and Role are not queryable or summary by default, but you can expose the following attributes:

- For User:
  - `MemberAdminGroups`
  - `adminRoles`
  - `adminGroupsRule`
- For Role and extensions of Role: `role_applications`

For attribute definitions, click the Display Schema button on the Debug Pages to view the `IDMObjectClass` schema. Administrators must have View rights for `IDMSchemaConfig` to view the `IDMObjectClass` schema.

Operational Attributes

Waveset predefines several attributes that are required for the repository to work correctly. `ID`, `type`, and `name` are especially important.

Every PersistentObject stored in the repository has a *globally unique internal identifier (ID)*. An ID value is unique across time and space, and a generated ID value is never re-used. (Some predefined Waveset objects have well known identifiers that are defined as program constants. These are known as *fake IDs*.) The repository ensures that an object's ID will never change.

Objects of the same type typically *map* to the same Java class. That is, they are constructed as instances of the same Java class when *deserialized*. Where there is not a one-to-one correspondence between type and Java class, every object of the same type at least uses the same mechanism to look up the corresponding Java class. (For example, some types of objects expose a `class` attribute that contains the fully qualified class name.)

An object's name must be unique within *type*. That is, only one object of a type can have a particular name. (However, another object of a different type can have the same name.) Thus, each type effectively defines a subordinate namespace. You can change an object's name, but you cannot change an object's ID.

View Attributes

A *view* is a collection of name/value pairs that are assembled from one or more objects stored in the repository, or read from resources. The value of a view attribute can be atomic, such as a string, a collection such as a list, or reference to another object.

Whenever you create or modify a user account from the Waveset Administrator or User Interfaces, you are indirectly working with the User view. Workflow processes also interact with the User view. When a request is passed to a workflow process, the attributes are sent to the process as a view. When a manual process is requested during a workflow process, the attributes in the user view can be displayed and modified further.

Working with views is extensively documented in the *Views* chapter of Deployment Reference.

Resource User Attributes

Resource User attributes map Waveset account attributes to resource account attributes in a schema map (right side). The list of attributes varies for each resource. You can remove unused attributes from the schema map page. However, adding attributes might require editing the adapter code.

The Resource User attributes are used only when the adapter communicates with the resource.

Working with Resource User attributes is extensively documented in Resource Reference.

Identity System User Attributes

Identity System User attributes define an internal Waveset value that corresponds to a Resource User attribute. The Identity System User attributes can be used in rules, forms, and other Waveset-specific functions. Waveset displays these attributes on the left side of the schema map.

Working with Identity System User attributes is extensively documented in Resource Reference.

Other Standard Attributes

You can use some of the other standard attributes to restrict access to objects (such as `MemberObjectGroups`, `subType`, or `authType`) or to represent historical information (such as the creator and date created.).

MemberObjectGroups

Every persistent object belongs to at least one object group. Each value of this multi-valued attribute is the ID of an ObjectGroup object.

ObjectGroups are exposed as Organizations in the Waveset Administrator and User Interfaces. ObjectGroup membership governs Session-level authorization (that is, administrator and user access to repository objects), but the repository itself ignores object group membership.

creator, createDate, lastModifier and lastModDate

These values record historical information about each object. These attributes are maintained (but are not used) by the repository.

PropertyList

Every persistent object can contain an arbitrary list of Properties. This feature is not widely used.

subType

Every persistent object can have a `subType` attribute. For example, Waveset uses Attribute.SUBTYPE to select separate lists of the available correlation rules and confirmation rules.

authType

The `authType` attribute allows fine-grain authorization to be performed (that is, access to be scoped or restricted) for users who do not control any organization (object group). These subjects would otherwise have no access in Waveset's standard authorization scheme.

Using Attribute Conditions

An *attribute condition* is an expression that tests the values of an attribute. Attribute conditions are commonly used to select the subset of objects that match certain criteria.

Each attribute condition expresses a single criterion and consists of:

- Attribute name (of a queryable attribute)

- Operator (a kind of check or comparison to be made)
- Operand (a specified set of values)

Attribute Condition Operators

AttributeCondition defines the following operators.

TABLE 1–1 Attribute Condition Operators

Operator	Description
`EQ, EQUALS`	Object has at least one value for the specified attribute that is lexically equal to (ignoring case) the operand.
`NE, NOT_EQUALS`	Object has no value for the specified attribute that is lexically equal to (ignoring case) the operand.
`GT, GREATER_THAN`	Object has at least one value for the specified attribute that is lexically greater than (ignoring case) the operand.
`GE`	Object has at least one value for the specified attribute that is lexically greater than or equal to (ignoring case) the operand.
`LE`	Object has at least one value for the specified attribute that is lexically less than or equal to (ignoring case) the operand.
`LT, LESS_THAN`	Object has at least one value for the specified attribute that is lexically less than (ignoring case) the operand.
`STARTS_WITH`	Object has at least one value for the specified attribute that is an initial substring (ignoring case) of the operand.
`ENDS_WITH`	Object has at least one value for the specified attribute that is a final substring (ignoring case) of the operand.
`CONTAINS`	Object has at least one value for the specified attribute that is a substring (ignoring case) of the operand.
`IS_PRESENT`	Object has at least one value for the specified attribute. (This operator takes no operand.)
`NOT_PRESENT`	Object has no value for the specified attribute. (This operator takes no operand.)
`IN, IS_ONE_OF`	Object has at least one value for the specified attribute that is lexically equal to (ignoring case) one of the values in the (list) operand.

Note – RelationalDataStore optimizes evaluation by translating each attribute condition into an appropriate predicate that becomes part of the `WHERE` clause for the operation. However, no special logic is required to handle multi-valued attributes. RelationalDataStore automatically generates appropriate SQL DML to handle this.

An attribute condition applies to *each value of an attribute*. (Specifically, operator `NE` is `true` if, and only if, an object has no value for the specified attribute that equals the specified operand. Operator `EQ` is `true` if an object has at least one value for the specified attribute that matches the specified operand.)

Implicitly `AND`ed

A set of attribute conditions is implicitly `AND`ed. This means that a set of attribute conditions evaluates to `true` if, and only if, every attribute condition in the set evaluates to `true`. Conversely, a set of attribute conditions evaluates to `false` as soon as any attribute condition in the set evaluates to `false`.

Waveset attribute conditions expose operators that are generally useful. Typically, you can express a set of selection criteria using Waveset attribute conditions. A few criteria cannot be expressed, but even these are often better addressed by adding (or changing the representation of) a queryable attribute.

Example Scenario: Populating Organizations with User Member Rules

You can use the following attributes to determine the set of users in a given organization:

- **External (to Waveset) resource account attributes**. In this case, you need both the resource account ID and the resource name (for example, `acctid:resname`) to find the matching Waveset user because more than one Waveset user might have the same `acctid` but on different resources.
- **Waveset user account attributes** (for example, name, location, manager).

To get the "or'ed" effect, do not use multiple attribute conditions. Instead, use the "is one of" operator with a list of operands, as follows:

```
<list>
  <new class='com.waveset.object.AttributeCondition'>
    <s>firstname</s>
    <s>is one of</s>
    <list>
      <s>Nicola</s>
      <s>Paolo</s>
    </list>
  </new>
</list>
```

Example Scenario: Including All Users Without Administrative Roles

You need a rule to include all users except those with specified administrative roles.

Because attribute conditions are implicitly `AND`ed together, you can use two attribute conditions:

- Condition that selects users with at least one admin role (which in effect excludes non-administrative users). This condition specifies that a matching user has at least one value for the `adminRoles` attribute.

  ```
  <AttributeCondition>
     <s>adminRoles</s>
     <s>exists</s>
   </AttributeCondition>
  ```
- Condition that excludes users with any of a set of specific admin roles. This condition specifies that no value of the `adminRoles` attribute is `ar1` or `ar2`.

  ```
  <AttributeCondition>
     <s>adminRoles</s>
     <s>is not</s>
     <list>
       <s>ar1</s>
       <s>ar2</s>
     </list>
   </AttributeCondition>
  ```

Taken together, these conditions specify that the user must have an admin role that is not in the specified list.

Using Secret Attributes

Waveset displays attribute values in clear text on the Results pages, even when you have set the attribute for display with asterisks in an Edit form. To prevent attribute valued from being displayed in the cache, you can register the attribute as *secret*. Secret attribute values are not displayed in clear text in the browser cache, but these attributes are processed by Waveset just like any other attribute.

For example, a social security number is an attribute that administrators typically register as a secret attribute.

When rendering the results table, Waveset checks to determine whether any of the attributes are registered as secret, and displays the values of secret attributes with asterisks only.

To register a secret attribute, add that attribute to the System Configuration object as follows:

```
<Attribute name='secretAttributes'>
   <List>
     <String>email</String>
     <String>myAttribute</String>
```

```
   </List>
</Attribute>
```

CHAPTER 2

Working with Authorization Types

This chapter presents a conceptual overview of authorization types (AuthTypes) as used in Waveset deployments. Topics in this chapter include

- "What are Authorization Types?" on page 27
- "How Waveset Uses Authorization Types" on page 28
- "Why Use Authorization Types?" on page 28
- "Using Authorization Types" on page 29
- "Authorization Types and Capabilities" on page 30
- "Creating an Authorization Type" on page 31

What are Authorization Types?

Waveset provides *authorization types* as a mechanism for assigning authorization rights to objects without requiring code changes. This extensible mechanism is independent of the repository storage type, and is especially useful for TaskDefinition and Configuration objects. Although these objects share the same repository type, each object type can perform different functions that consequently require different authorization policies. For example, rules must have an authorization type of UserMembersRule to appear in the User Members Rules drop-down list. Both default and custom authorization types reside in the Configuration:AuthorizationTypes object.

Authorization types are *repository-type independent*, which means that you can define one authorization type and assign it to, for example, both Configuration and Rule objects. This allows you to use authorization types to filter lists of objects of a single type, or as a means of granting access to a related set of objects to a subset of Waveset administrators with a specific capability.

How Waveset Uses Authorization Types

Waveset uses authorization types during access checks when comparing the caller's capabilities against an object's authorization type. When an authorization type extends an existing repository type, access control follows the implied 'inheritance' change. Specifically, if an administrator has rights on the parent type, he has the same rights on the child type. However, if an administrator has rights on the child type, but no rights on the parent, then the administrator can access objects of the child type only.

For example, consider the following authorization types, administrators and objects:

Authorization settings:

```
Configuration   (repository type)
<AuthType name='Fruit' extends='Configuration'/>
<AuthType name='Vegetable' extends='Configuration'/>
```

Rights are assigned as follows:

AdminA (has Right.VIEW on Configuration)

AdminB (has Right.VIEW on Fruit)

AdminC (has Right.VIEW on Vegetable)

ObjectA of type Configuration, no authtype

ObjectB of type Configuration, authtype == Fruit

ObjectC of type Configuration, authtype == Vegetable

The preceding authorization settings determine the following access privileges on the specified objects:

- AdminA can view ObjectA, ObjectB and ObjectC
- AdminB can view ObjectB only
- AdminC can view ObjectC only

Why Use Authorization Types?

You use authorization types within your deployment to accomplish the following:

- Restrict a list of a single type of objects to those specific for a purpose (very similar to SubType). For example, `<AuthType name='foo' extends='moo'/>`
- Group objects of different types to make them available to a specific class of administrators. For example, `<AuthType name='foo' extends='red,green,blue'/>`

Using Authorization Types

Authorization Type (`AuthType`) objects are members of the `Configuration:AuthorizationTypes` objects. An AuthType object can be assigned directly by an `authType` attribute member of a persistent object or indirectly through an AdminGroup (described below).

Configuration:AuthorizationTypes Object

The `Configuration:AuthorizationTypes` object defines valid authorization types. Each authorization type is declared in an `<AuthType>` element:

```
<AuthType name='SPML' extends='Configuration'/>
```

The `AuthTypes` element contains a list of `AuthType` elements. Each `AuthType` has, at minimum, a `name` attribute and typically an `extends` attribute. The value of the `extends` attribute must be the name of another authorization type or repository type.

You can add or remove authorization types by modifying this object. You can modify the `Configuration:AuthorizationTypes` only by editing its XML representation.

AuthType Element

This element requires the `name` property. The example below displays the correct syntax for an `<AuthType>` element. The following example shows how to add a custom task to move multiple users into a new organization.

```
<Configuration name='AuthorizationTypes'>
   <Extension>
      <AuthTypes>
         <AuthType name='Move User' extends='TaskDefinition,TaskInstance,TaskTemplate'/>
      </AuthTypes>
   </Extension>
</Configuration>
```

The `AuthType` element supports the following attributes.

TABLE 2–1 `AuthType` Attributes

AuthType Object Attributes	Description
`name`	Identifies the authorization type.
`extends`	Specifies the name of one or more authorization types or repository types that become the supertype of this type. Rights granted by the supertype are inherited.
`displayName`	Provides an alternate display name for this type, typically a message catalog key.

TABLE 2–1 `AuthType` Attributes *(Continued)*

AuthType Object Attributes	Description
`auditKey`	Identifies the audit log key to be used for audit records associated with objects of this type. If none is specified, the audit key of the base type is used.
`allowedRights`	Provides a comma delimited list of right names. This defines the rights that can be used with this authorization type in a permission definition. If not specified, all rights are allowed.

Authorization Types and Capabilities

Authorization types are a key component of the End User authorization model. With authorization types, you can define capabilities, or `AdminGroups`, and then assign those capabilities to users.

AdminGroups Objects

After defining an authorization type, you can reference it in the Permission objects stored within `AdminGroup` objects. The following XML example defines an `AdminGroup` (called a *capability*) that you can assign to a user.

```
<AdminGroup name='EndUser'>
   <Permissions>
       <Permission type='EndUserTask' rights='View'/>
       <Permission type='EndUserRule' rights='View'/>
   </Permissions>
 <MemberObjectGroups>
 <ObjectRef type='ObjectGroup' id='#ID#All' name='All'/>
 </MemberObjectGroups>
</AdminGroup>
```

In this example, the two `Permission` elements both use type names that are authorization types rather than repository types. Only `TaskDefinition` objects that are assigned an `EndUserTask` authorization type will be accessible to a user that holds this capability. (A *capability* conveys set of rights to one or more authorization types or repository types.) Because authorization types are essentially hierarchical with other authorization types and repository types, having rights on a parent in the “type hierarchy” grants the same rights to all children.

EndUser Capability

You can use the `AdminGroup EndUser` capability to assign permissions to non-administrative users that typically do not have assigned capabilities and do not control any organizations. The default definition of this capability was given in the example in the Permission Extensions section.

Waveset implicitly assigns all users the `EndUser` capability. This capability permits users to view several types of objects, including tasks, rules, roles, and resources. Although you can assign capabilities to end users, you may prefer not to. Waveset defines a user with explicitly assigned capabilities as an administrator, and the system caches information about administrators that results in an effective upper limit on the number of administrators an installation can have.

You can use the `EndUserLibrary` authorization type. The `EndUser` capability (or `AdminGroup`) has List and View access to Libraries with the `EndUserLibrary` authType.

To give users access to the contents of a Library, set `authType='EndUserLibrary'` and ensure that the Library's `MemberObjectGroup` is set to `All`.

Creating an Authorization Type

You can create a new authorization type by extending the existing `TaskDefinition`, `TaskInstance`, and `TaskTemplate` authorization types. You can use one of the following methods to add an authorization type:

- Create a new authorization type using the `<AuthType>` element.
- Edit the Authorization Types Configuration object in the repository by adding the new authorization type element (`AuthType`) for your task.

Assigning an Authorization Type to a Repository

By setting an authorization type on a respiratory, you can restrict which users can see, modify, or delete particular object types. To define an authorization type for a repository type, set the authorization type name to the name of a repository type and omit the `extends` attribute.

Example: Setting End-User Authorization Types

Waveset implements the User Admin role and assigns it to all users by default. This role encapsulates the `EndUser AdminGroup` that provides two end-user authorization types (AuthTypes) and several list permissions for various object types.

These end-user authorization types include:

- **EndUserRule**. Allows access to rule objects that have the `EndUserRule` AuthType specified in the object, as follows:

  ```
  <Rule authType='EndUserRule' ...>
  ```
- **EndUserTask**. Allows access to TaskDefinition objects that have the `EndUserTask` AuthType specified in the object, as follows:

  ```
  <TaskDefinition authType='EndUserTask' ...>
  ```

- **EndUserLibrary**. Allows access to the contents of a Library object.

  To implement this AuthType, set the AuthType to `EndUserLibrary` and ensure the Library's `MemberObjectGroup` is `All`. (The `EndUser` capability (`AdminGroup`) has List and View access to Libraries whose authorization type is `EndUserLibrary`.)

Example: Using Authorization Types to Restrict Visibility on Resources

You can use authorization types to restrict visibility on resources on the resource level. Rather than move resources into special organizations, you can

- Define an authorization type for each resource (for example, Resource-Corporate-LDAP)
- Build capabilities with rights for those resource types

When assigning capabilities to users, do not assign a capability that includes rights to a generic resource type. Instead, assign users a capability with rights for a specific resource type.

Note – For an example of stock authorization types defined in the system, see the `admingroups.xml` file.

▼ To Define a Resource-Specific Authorization Type

1 **Add an entry to `Configuration:AuthorizationTypes` object.**

```
<AuthType name='Resource-Corporate-LDAP' extends='Resource'/>
```

2 **Derive a variant of one of the standard capabilities, such as `Resource Administrator`. Note that the only difference between this capability and the standard `AdminGroup` is the type name in the `Permission`, which is `Resource-Corporate-LDAP` instead of `Resource`.**

```
<AdminGroup name='Corporate LDAP Resource Administrator'
           protected='true'
           displayName='UI_ADMINGROUP_RESOURCE_ADMIN'
           description='UI_ADMINGROUP_RESOURCE_ADMIN_DESCRIPTION'>
   <AdminGroups>
       <ObjectRef type='AdminGroup' id='#ID#Resource Group Administrator'/>
       <ObjectRef type='AdminGroup' id='#ID#Resource Report Administrator'/>
       <ObjectRef type='AdminGroup' id='#ID#Connect Organizations'/>
       <ObjectRef type='AdminGroup' id='#ID#Connect Policies'/>
   </AdminGroups>
   <Permissions>
       <Permission type='AttributeDefinition' rights='View'/>
       <Permission type='Resource-Corporate-LDAP' rights='View,List,Create,Modify,Delete,Execute'/>
       <Permission type='ResourceUIConfig' rights='Create,Modify'/>
       <Permission type='Rule' rights='View'/>
       <Permission type='User' rights='View,List'/>
   </Permissions>
   <MemberObjectGroups>
       <ObjectRef type='ObjectGroup' id='#ID#All' name='All'/>
   </MemberObjectGroups>
</AdminGroup>
<ObjectRef type='AdminGroup' id='#ID#Connect Resource Groups'/>
```

Example: Granting Access to a Specific Part of Waveset

You can also use authorization types to grant fine-grained administrative control of a very specific part of Waveset to a set of users.

You create an AuthType, assign objects to that AuthType, and then create a capability that grants that AuthType. When you assign this capability to a set of users, they can only see the area of the system that the authorization type and capability allow them to see.

The following example assigns the `LimitedReportType` authorization type to a `TaskDefinition`, and the `Run Limited Report` capability to a user. Consequently, that user can only execute reports where `TaskDefinition` is the `LimitedReportType` authorization type.

```
<AuthType name='LimitedReportType' extends='TaskDefinition'/>
<AuthType name='LimitedReportType' extends='TaskInstance'/>
<AdminGroup name='Run Limited Report' ...>
...
<Permissions type='LimitedReportType' rights='View,Execute'/>
 ...
</AdminGroup>
```

◆ ◆ ◆ CHAPTER 3

Data Loading and Synchronization

This chapter presents an overview of the techniques that can be used to load and synchronize account information from a resource into Waveset.

It is important to clearly distinguish between Waveset users and resource accounts. The following definitions make it easier to understand the topic:

- **User**. A virtual identity that is managed by Waveset. An Waveset user may refer to any number of accounts.
- **Account**. A concrete identity that is managed by a resource (or, more precisely, by an external system or application that is represented as a Resource object in Waveset). For example, an entry in `/etc/passwd` on a UNIX system, an entry in the SAM database on a Windows system, and a UserProfile in RACF all represent accounts.
- **Administrator**. A person with responsibility for configuring and maintaining the Waveset system.

Waveset stores information about known resource accounts and users in the *account index*. At a minimum, each entry in the account index contains an account ID and an Waveset resource ID. An entry might also contain additional information, such as the native GUID or the status (enabled/disabled) of an account. An entry might also record the ID of an Waveset user as the owner of the account, or it might record a list of possible owners.

Topics discussed in this chapter include:

- “Types of Data Loading” on page 36
- “Load Operation Context” on page 41
- “Managing Reconciliation” on page 42
- “Managing Active Sync” on page 49

Types of Data Loading

Data loading is the process of importing account information from resources into Waveset and assigning these accounts to Waveset users. Waveset supports the following features that load account data from resources:

- Discovery. Provides basic functions that initially load resource accounts into Waveset.
- Reconciliation. Periodically loads resource account information into Waveset, taking action on each account according to configured policy.
- Active Sync. Allows information that is stored in an "authoritative" external resource (such as an application or database) to synchronize with Waveset user data. An Active Sync-enabled adapter "listens" or polls for changes to the authoritative resource.

Each of these concepts is discussed in detail. A table comparing the types of data loading can be found in *Summary of Data Loading Types*.

Discovery

The discovery processes are designed to be used when a resource is being deployed for the first time. They provide a means to load account information into Waveset quickly. As a result, they do not provide all the features found in reconciliation or Active Sync. For example, the discovery process does not add entries to the Account Index. Nor can you run workflows before or after discovery. However, the discovery processes allow you to determine more quickly whether correlation rules are working as expected.

When you begin a discovery process, Waveset determines whether an input account matches (or correlates with) an existing user. If it does, the discovery process merges the account into the user. The process will create a new Waveset user from any input account that does not match.

Waveset provides the following discovery functions:

- Load From File. Reads accounts listed in a file and loads them into Waveset.
- Load From Resource. Extracts accounts from a resource and loads them directly into Waveset.
- Create Bulk Action. Executes user creation commands listed in a file.

See the following sections for more information about these discovery processes.

Load from File

The Load from File discovery process reads account information that has been written into an XML or CSV (comma-separated values) file.

Some resources, such as Active Directory, have the ability to export native account information into a comma-separated values (CSV) format. These CSV files can be used to create Waveset accounts. See Business Administrator's Guide for more information about CSV formatting.

When you load from a file, you must specify which account correlation and confirmation rules to use. See *Correlation and Confirmation Rules* for more information.

Load from Resource

The Load from Resource feature scans a target system and returns information on all users. Waveset then creates and updates users. An adapter must have been configured for the resource before you can load from the resource.

When you load from a resource, you must specify which account correlation and confirmation rules to use. See *Correlation and Confirmation Rules* for more information.

Create Bulk Action

Bulk actions allow you to act on multiple accounts at the same time. You can use bulk actions to create, update, and delete Waveset and resource accounts, but this discussion will be limited to Waveset creating accounts. See Business Administrator's Guide for a full description of bulk actions.

Bulk actions are specified using comma-separated values (CSV). The structure of these values differs from those specified in a Load from File process.

The CSV format consists of two or more input lines. Each line consists of a list of values separated by commas. The first line contains field names. The remaining lines each correspond to an action to be performed on an Waveset user, the user's resource accounts, or both. Each line should contain the same number of values. Empty values will leave the corresponding field value unchanged.

Two fields are required in any bulk action CSV input:

- **user**. Contains the name of the Waveset user.
- **command**. Contains the action taken on the Waveset user. For creating Waveset users, this value must be Create.

The third and subsequent fields are from the User view. The field names used are the path expressions for the attributes in the views. See *Understanding the User View* in Deployment Reference for information on the attributes that are available in the User View. If you are using a customized User Form, then the field names in the form contain some of the path expressions that you can use.

Following is a list of some of the more common path expressions used in bulk actions:

- `waveset.roles`. A list of one or more role names to assign to the Waveset account.
- `waveset.resources`. A list of one or more resource names to assign to the Waveset account.
- `waveset.applications`. A list of one or more resource groups to assign to the Waveset account.

- `waveset.organization`. The organization name in which to place the Waveset account.
- `accounts`[*resource_name*].*attribute_name*. A resource account attribute. The names of the attributes are listed in the schema for the resource.

Some fields can have multiple values. For example, the `waveset.resources` field can be used to assign multiple resources to a user. You can use the vertical bar (|) character (also known as the "pipe" character), to separate multiple values in a field. The syntax for multiple values can be specified like this:

```
value0 | value1 [ | value2 ... ]
```

The following example illustrates Create bulk actions:

```
command,user,waveset.resources,password.password,password.confirmPassword,accounts[AD].description
,accounts[Solaris].comment
Create,John Doe,AD|Solaris,changeit,changeit,John Doe,John Doe
Create,Jane Smith,AD,changeit,changeit,Jane Smith,
```

The Create bulk action is more versatile than the from Load from File process. Bulk actions can work with multiple resources, while Load from File loads information from one resource at a time.

Reconciliation

Reconciliation compares the contents of the account index to what each resource currently contains. Reconciliation can perform the following functions:

- Detect new and deleted accounts
- Detect changes in account attribute values
- Correlate accounts with Waveset users
- Detect accounts that are not associated with Waveset users
- Run a workflow in response to each account situation that it detects
- Detect when a user has been moved from one container on a resource to another container on a resource

---

Note – An adapter must have been configured for the resource before you can reconcile. See Resource Reference for more information about adapters.

There are two types of reconciliation: full and incremental.

Full Reconciliation

Full reconciliation recalculates the existence, ownership, and situation for each account ID listed by the adapter. It examines each Waveset user that claims the resource to recalculate ownership.

An Waveset user can claim a resource by:

- Having a role that implies the resource
- Having a direct resource assignment
- Referring to an account on that resource
- Having a resource group

For each account, reconciliation process confirms that any Waveset owner recorded in the Account Index still exists and still claims the account. Any account that does not have an owner is correlated with Waveset users (as long as reconciliation policy for that resource specifies a correlation rule). If a correlation rule suggests one or more possible owners, then each of them will be double-checked in a confirmation rule (if one is specified). See "Correlation and Confirmation Rules" on page 43for more information about rules.

Once a situation has been determined for the account, reconciliation will perform any response that is configured in the reconciliation policy for that resource. If the reconciliation policy specifies a workflow to be performed per-account, full reconciliation will perform this for each account that is reconciled, after the situation action is performed. See "Reconciliation Workflows" on page 45 for more information about workflows.

Incremental Reconciliation

Incremental reconciliation is analogous to incremental backup: it is faster than full reconciliation, and does most of what you need, but is not as complete as full reconciliation.

Incremental reconciliation trusts that the information maintained in the account index is correct. Trusting that the list of known account IDs is correct, and that ownership of the account by any Waveset owner is correctly recorded, allows incremental reconciliation to skip or shorten several processing phases.

Incremental reconciliation skips the step of examining Waveset users that claim the resource. Incremental reconciliation also calculates a situation only for accounts that have been added or deleted since the resource was last reconciled. It does this by comparing the list of account IDs in the account index for that resource to the list of account IDs returned by the resource adapter. New accounts are recorded as existing, deleted accounts are recorded as no longer existing, and only these two sets of accounts are processed further.

Because incremental reconciliation is much faster and uses fewer processing cycles than full reconciliation, you may want to schedule incremental reconciliation more frequently and schedule full reconciliation less often.

Active Sync

Active Sync "listens" or polls for changes to a resource, detecting incremental changes in real time. Because Active Sync is designed to detect changes, it should not be used to load account information into Waveset for the first time. Instead, use reconciliation or a discovery process.

In general, you run reconciliation on an Active Sync resource in the following circumstances:

- To perform an initial load on the resource.
- To detect any attributes that have not been updated in Waveset because Active Sync has been configured to ignore or filter out the attributes.

Active Sync differs from reconciliation in the following ways:

- Active Sync allows an administrator to specify a user form that ensures attributes across multiple accounts are kept synchronized.
- A process rule can be implemented that fully controls all Active Sync processing. This is typically enabled when extraordinary actions need to be performed when an account on a resource changes, such as editing multiple objects in the repository.

Active Sync requires the use of an Active Sync-enabled adapter that has been properly configured. See Business Administrator's Guide for more information about configuring a resource to implement Active Sync.

Summary of Data Loading Types

The following table compares the capabilities of discovery and reconciliation.

TABLE 3–1 Summary of Data Loading Types

Function	Discovery	Reconciliation	Active Sync
Detect new accounts	Yes	Yes	Yes
Detect deleted accounts	No	Yes	Yes
Detect changes in account attribute values	No	Yes	Yes
Detect accounts that are not associated with Waveset users	Yes	Yes	Yes
Detect when a user has been moved from one container on a resource to another container on a resource	No	Yes	Yes
Correlate accounts with Waveset users	Yes	Yes	Yes
Run a workflow in response to each account situation that it detects	No	Yes	Yes
Can be scheduled	No	Yes	Yes

TABLE 3–1 Summary of Data Loading Types (Continued)

Function	Discovery	Reconciliation	Active Sync
Incremental mode	No	Yes	Not applicable
Add entries to the account index	No	Yes	Yes
Synchronize attributes on multiple resources	No	No	Yes

Load Operation Context

The following table provides information about the common Waveset processes or tasks related to the load operations category.

TABLE 3–2 Load Operations Processes/Tasks

Process or Task	How it is Used	Namespace
Load from File	Retrieves account information from a CVS or XML file (invoked through Administrator Interface). Waveset reads a `WSUser` object from a file, converts it to the User view, and applies the form. The attributes are processed as if they were extended attributes of the Waveset user. Attributes are put in `accounts[Lighthouse]` and will only be put under the `global` attribute if the form defines global fields for each of them.	All attribute values for each line in the file are pulled into the global namespace: `global.`<*attr name*> **Note**: Applies to `create` operations only.
Load from Resource	Retrieves account information from a particular resource (invoked through Administrator Interface and uses an adapter to list and fetch accounts).	All attribute values for each account on the resource are pulled into the global namespace. `global.`<*LHS Attr Name*> **Note**: Applies to `create` operations only.
Bulk Operations	Retrieves commands and User view data from a CVS file (invoked through Administrator Interface). You can specify any attribute in the User view namespace. Attribute names are specified using the view path syntax. See "Understanding the User View" in *Oracle Waveset 8.1.1 Deployment Reference* for more information about the User view namespace and view path syntax	Attribute values from the file are pulled into the global namespace: ■ `accounts[*].*` ■ `waveset.*` ■ `accountInfo.*` ■ `global.*` **Note**: There is no authorized session available.

Managing Reconciliation

The reconciliation process is primarily managed through the Administrator Interface. However, there are some aspects of reconciliation that cannot be accomplished from this interface. For example, you might need to create new correlation and confirmation rules, reconciliation workflows, or edit the Reconcile configuration object. The following sections describe these features, and others. For general information about defining reconciliation policy, see Business Administrator's Guide.

Reconciliation Policy

Reconciliation policies allow you to establish a set of responses, by resource, for each reconciliation task. Within a policy, you select the server to run reconciliation, determine how often and when reconciliation takes place, and set responses to each situation encountered during reconciliation. You can also configure reconciliation to detect changes made natively (not made through Waveset) to account attributes.

Each of these policy settings can be defined at several scopes:

- Globally (for all resource types)
- For a specific resource type
- For an individual resource instance

The value at each scope becomes the default for each sub-scope. Thus, reconciliation policy defines an *inheritance tree*:

- The global value becomes the default for every resource type.
- Each resource type can inherit the global value or specify a value.
- The resource type value is the default for every resource instance of that type.
- Each resource instance can inherit value of its parent resource type, or specify a value.

Inheritance makes it easier to manage policy for a large number of resources (especially if many of them will have the same settings).

For example, if you want to treat all resources in the same way, you need to manage only one set of policy settings, at the global level. If you want to treat all Windows resources one way and all Solaris resources another way, you need to manage policy settings at only two scopes: one for each of these two resource types. If there are exceptions to the policy defined at the resource type level for a few specific resource instances, the necessary policy settings can be overridden (specified) for those individual resources. Since each policy setting is inherited separately, only the settings that differ need to be specified; the other policy settings may still inherit their values from above.

Correlation and Confirmation Rules

Waveset matches resource accounts that are not linked to a user with Waveset users in two phases:

- **Correlation**. Finding potential owners
- **Confirmation**. Testing each potential owner

A *correlation rule* looks for Waveset users that might own an account. A *confirmation rule* tests an Waveset user against an account to determine whether the user actually does own the account. This two-stage approach allows Waveset to optimize correlation, by quickly finding possible owners (based on name or attributes), and by performing expensive checks only on the possible owners.

Reconciliation policy allows you to select a correlation rule and a confirmation rule for each resource. (You may also specify No Confirmation Rule.) The default correlation rule is to look for a user with a name that exactly matches the account ID of the input account. By default, no confirmation rule is used.

Note – Correlation and confirmation rules are also used for discovery and Active Sync.

Waveset predefines a number of correlation and confirmation rules in `sample/reconRules.xml`. You can also write your own correlation and confirmation rules. Any rule object with a subtype of `SUBTYPE_ACCOUNT_CORRELATION_RULE` or `SUBTYPE_ACCOUNT_CONFIRMATION_RULE` automatically appears in the appropriate Reconciliation Policy selection list.

Correlation Rules

A correlation rule can generate a list of user names based on values of the attributes of the resource account. A correlation rule may also generate a list of attribute conditions (referring to queryable attributes of a user object) that will be used to select users.

A correlation rule is run once for each unclaimed account.

Note – A correlation rule should be relatively “inexpensive” but as selective as possible. If possible, defer expensive processing to a confirmation rule.

Waveset predefines several correlation rules in `sample/reconRules.xml`:

- **User Name Matches AccountId**. Returns the value of the `accountId` attribute. It selects as a possible owner any Waveset user with a name that matches the resource account ID. This is the default correlation rule.
- **User Owns Matching AccountId**. Returns a list of attribute conditions. This will select as a possible owner any Waveset user that owns a resource account that matches the same `accountId` value.
- **User Email Matches Account Email**. Returns a list of attribute conditions that will select Waveset users based on the account's `email` attribute.

Input for any correlation rule is a map of the account attributes. Output must be one of:

- String (containing user name or ID)
- List of String elements (each a user name or ID)
- List of WSAttribute elements
- List of AttributeCondition elements

A more complicated rule might combine or manipulate account attribute values to generate a list of names or a list of attribute conditions.

Note – Attribute conditions must refer to queryable attributes, which are configured as `QueryableAttrNames` in the `UserUIConfig` object.

For example, `reconRules.xml` contains a fourth sample correlation rule, User FullName Matches Account FullName. XML comments disable this rule, because it will not work correctly without additional configuration. This rule looks for Waveset users based on `fullname`, but this attribute is not queryable by default.

Correlating on an extended attribute requires special configuration:

- The extended attribute must be specified as queryable in `UserUIConfig` (added to the list of `QueryableAttrNames`).
- The Waveset application (or the application server) may need to be restarted for the `UserUIConfig` change to take effect.

Confirmation Rules

A confirmation rule is run once for each matching user returned by the correlation rule.

A typical confirmation rule compares internal values from the user view to the values of account attributes. As an optional second stage in correlation processing, the confirmation rule performs checks that cannot be expressed in a correlation rule (or that are too expensive to evaluate in a correlation rule). In general, you need a confirmation rule only in the following circumstances:

- The correlation rule may return more than one matching user
- User values that must be compared are not queryable

Waveset predefines two confirmation rules in `sample/reconRules.xml`:

- **User Email Matches Account Email**. Returns a value of true if the user's email matches the account's email. This illustrates the fact that many ownership decisions could be expressed with either a correlation rule or a confirmation rule. However, since the `email` attribute of an Waveset user is automatically queryable, it would almost always be more efficient to express this as a correlation rule.
- **User First And Last Names Match Account**. Uses the XPRESS language to compare the user's first and last name to the same values of the account.

Inputs to any confirmation rule are:

- **`userview`**. Full view of an Waveset user.
- **`account`**. Map of resource account attributes.

A confirmation rule returns a string-form Boolean value of true if the user owns the account; otherwise, it returns a value of false.

The default confirmation rule is No Confirmation Rule. This assumes that the correlation rule is selective enough to find at most one user for each account. If the correlation rule selects more than one user, the account situation will be DISPUTED.

CorrelationPlan Objects

Correlation logic can indicate a resource account's type. During reconciliation, the automatic link must know about account type because no form is used to perform this action.

When reconciliation performs a LINK response for a resource account, it typically assigns the account to the user as the default account type. However, on a resource that is configured for multiple accounts per user, this may not always be appropriate. Specifically, discovered accounts can belong to a specific account type and should be linked to the user as such. To assign the appropriate account type, reconciliation must be informed of the account type to use. Waveset accomplishes by returning this information as part of the result of the correlation rule.

A CorrelationPlan object extends the result of a correlation rule to allow the account type to be identified. Therefore, a correlation rule must return a CorrelationPlan if the account is of a specific account type. However, a CorrelationPlan can also be used for standard resources as well. Unless specifically set, a CorrelationPlan indicates the default account type.

Refer to `sample/correlationRules.xml` and the Javadoc for examples and details on using a CorrelationPlan as the result of a correlation rule.

Reconciliation Workflows

You can extend typical reconciliation processing by exposing a number of attachment points for user-defined workflows.

If you are using expensive (that is, long running) per-account workflows, consider modifying your default workflow configuration as described in the *Configuring Workflow Properties* section of Deployment Reference.

Pre-Resource Workflow

A pre-resource workflow can be launched before any other reconciliation processing is started. The Notify Reconcile Start workflow is an example of a pre-resource workflow.

The Notify Reconcile Start workflow e-mails an administrator with notice that a reconcile has started for a resource. You must configure the Notify Reconcile Start e-mail template before running this workflow.

The following parameters are passed to the pre-resource workflow:

- **`resourceName`**. Name of the resource that will be reconciled.
- **`resourceId`**. Object ID of the resource that will be reconciled.

Per-Account Workflow

The per-account workflow is launched for each account processed by reconciliation, after the response (if any) has completed. The type of response and the response result do not affect this workflow.

The Notify Reconcile Response workflow is an example of a per-account workflow. It e-mails an administrator when the reconcile process attempts to automatically respond to a discovered situation. You must configure the Notify Reconcile Response e-mail template before running this workflow.

The following parameters are passed to the per-account workflow:

- **`accountId`**. Name of the resource account that was reconciled.
- **`resourceId`**. Object ID of the resource being reconciled.
- **`resourceName`**. Name of the resource being reconciled.
- **`userID`**. Object ID of the Waveset user identified as the account owner (by claim or correlation, depending on the situation). If no user is associated with the account, this is null.
- **`userName`**. Name of the Waveset user identified as the account owner (by claim or correlation, depending on the situation). If no user is associated with the account, this is null.
- **`initialSituation`**. The situation that was initially discovered for the account, triggering the response. The value is a valid message key.
- **`responseSuccess`**. Boolean value indicating whether the response completed successfully. If no response was performed, this is null.

- **`finalSituation`**. Reconciliation situation the account was in after applying the response. If the account no longer exists and Waveset contains no references to it, this is null.

Post-Resource Workflow

A post-resource workflow can be launched after all other reconciliation processing is complete. The Notify Reconcile Finish workflow is an example post-resource workflow.

The Notify Reconcile Finish workflow e-mails an administrator with notice that a reconcile has finished for a resource. You must configure the Notify Reconcile Finish e-mail template before running this workflow.

The following parameters are passed into the post-resource workflow:

- **`resourceName`**. Name of the resource that was reconciled.
- **`resourceId`**. Object ID of the resource just reconciled.

Auditing Native Changes

The Audit Native Change To Account Attributes workflow is launched when reconciliation or the provisioner detects a change to the attributes of a resource account that was not initiated through Waveset. Only user-specified attributes are monitored for changes. By default, no attributes are monitored.

The following parameters are passed to the workflow:

- **`resource`**. Resource object where the account was changed natively.
- **`accountID`**. Name of the resource account that was changed natively.
- **`prevAttributes`**. Map containing the monitored resource account attributes recorded by Waveset.
- **`newAttributes`**. Map containing the monitored resource account attributes currently set on the resource.
- **`attributeChanges`**. Map containing the List of generic objects that indicate which attributes have changed. Each object contains the previous and new values.
- **`formattedChanges`**. String representing the attribute changes in compact format, suitable for an audit record.

To audit native changes, you must do the following:

- On the **Edit Reconciliation Policy** page, select the **Detect native changes to account attributes** option from the **Attribute-level reconciliation** drop-down menu. You might need to uncheck the **Inherit resource type policy** check box to display a list of attributes. Select the attributes to audit.
- Add **Changes Outside Waveset** to the list of audit events. To do this, select the Configure tab, then Audit Events on the left.

Resource Scheduling

Reconciliation maintains two separate schedules for each resource: one for incremental reconciliation, and another for full reconciliation.

Each resource is scheduled by a separate “requester” task. Configuring a reconciliation schedule for a resource in the reconciliation policy GUI configures the TaskSchedule for the requester task. This allows reconciliation to be controlled by an external task scheduler, if desired. It also minimizes the overhead of the reconciliation task. A reconciliation daemon that is not doing anything consumes very few resources, since it periodically polls an in-memory queue (rather than querying the database for resources that are ready to reconcile).

Reconciliation accesses each resource through a resource adapter. Reconciliation calls the adapter *directly* to list accounts, iterate accounts, or fetch an individual resource account. Reconciliation also accesses resources *indirectly* through Provisioner, and uses Provisioner to create a resource account or Waveset user from a resource account.

Reconciliation and Provisioner both maintain the account index. Also, reconciling a resource prunes the Account Index each time. The reconciliation task automatically removes any entry for an account that no longer exists on the resource, unless that account is owned by an Waveset user. Therefore, it should not be necessary to attempt to manually clear the Account Index for a resource.

Each Waveset server runs reconciliation as a daemon task. This means that the Waveset scheduler starts the reconciliation task immediately and automatically restarts the task if it dies.

Note – Resource reconciliation is not automatically restarted. The `ReconTask` daemon itself is automatically restarted, which enables it to respond to any new request; but any request in process when the host server dies (or when the application server is shut down) is lost. The daemon does not restart resource reconciliation because it may be inappropriate to reconcile the resource at a time other than when requested.

Reconcile Configuration Object

The ReconcileConfiguration object contains several attributes that cannot be edited from the Edit Reconciliation Policy page.

The following table defines the reconciliation attributes. Use the debug pages to edit the attribute values in the ReconcileConfiguration object (#ID#Configuration:ReconcileConfiguration)

TABLE 3–3 Primary Attributes of ReconcileConfiguration Object

Attribute	Description
`fetchTimeout`	The number of milliseconds the reconciliation process should wait for a response from a resource when fetching an account. The default value is 1 minute (60000 milliseconds).
`listTimeout`	The number of milliseconds the reconciliation process should wait for a response from a resource when listing accounts. The default value is 10 minutes (600000 milliseconds).
`maxConcurrentResources`	The maximum number of resources that a server should reconcile concurrently. The default value is 3.
`maxQueueSize`	The maximum number of entries in a reconciliation server's work queue. The default value is 1000.

The following example shows the default values for the ReconcileConfiguration object.

EXAMPLE 3–1 Default Values for the ReconcileConfiguration Object

```
<Configuration id='#ID#Configuration:ReconcileConfiguration' name='Reconcile Configuration'>
   <Extension>
      <Object>
         <Attribute name='listTimeout' value='600000' />
         <Attribute name='fetchTimeout' value='60000' />
         <Attribute name='maxConcurrentResources' value='3' />
         <Attribute name='maxQueueSize' value='1000' />
      </Object>
   </Extension>
   <MemberObjectGroups>
      <ObjectRef type='ObjectGroup' id='#ID#All' name='All'/>
   </MemberObjectGroups>
</Configuration>
```

Managing Active Sync

Active Sync-enabled adapters can be managed in the Administrator Interface. This interface contains a wizard that allows an administrator to fully configure most aspects of Active Sync on a single adapter. The wizard also allows the administrator to construct a resource, or input, form, without using the Oracle Waveset Integrated Development Environment. For more details about the Active Sync wizard, see Business Administrator's Guide.

How Active Sync-Enabled Adapters Work

This section describes:

- Overview of the basic steps of adapter processing
- Active Sync variable context
- Using rules

- Using forms
- Launching workflow processes

Basic Steps of Adapter Processing

All Active Sync-enabled adapters follow the following basic steps when listening or polling for changes to the resource defined in Waveset. 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 Waveset object is affected.
3. Builds a map of user attributes to pass to the system, along with a reference to the adapter and a map of any additional options, which creates an Identity Application Programming Interface (`IAPI`) object.
4. Submits the `IAPI` object to the `ActiveSync` Manager.
5. `ActiveSync` Manager processes the object and returns to the adapter a `WavesetResult` object that informs the Active Sync-enabled adapter if the operation succeeds. This object can contain many results from the various steps that the Waveset system uses to update the identity. Typically, a workflow also handles errors within Waveset, often ending up as an Approval for a managing administrator.

Active Sync Namespace

The following table provides information about the common Waveset processes or tasks related to the Active Sync category.

Process or Task Running	How it is Used	Namespace
ActiveSync `IAPIUser`	■ Processes user-related changes on a particular resource. ■ Performs actions directly on the full User view before launching the designated workflow process.	Merges attributes from the ActiveSync event into the User view. Typical attributes on the Input Form include: ■ `accounts[*].*` ■ `waveset.*` ■ `accountInfo.*` ■ `activeSync.`<*LHS Attr Name*> ■ `activeSync.`*resourceName* ■ `activeSync.resourceId` ■ `activeSync.resource` ■ `display.session` (session for Proxy Admin) ■ `global.`<*LHS Attr Name*> (if `set globals` flag is set on resource)

Process or Task Running	How it is Used	Namespace
ActiveSync `IAPIProcess`	■ Processes generic events on a resource by creating a Process view. ■ Top-level fields in Process view are arbitrary inputs to the task. ■ Collects attributes related to launching the task under the `global` attribute. ■ Writes the workflow to retrieve inputs from under `global` rather than as top-level attributes.	Launches the specified task with ActiveSync poll attributes dumped into top-level workflow `global` attribute. Workflow attributes assume the form:`global.`<*LHS Attr Name*>

Using Rules

When the Active Sync-enabled adapter detects a change to an account on a resource, it either maps the incoming attributes to an Waveset user, or creates an Waveset user account if none can be matched and if the Active Sync resource has been configured to do so.

The Active Sync wizard allows you to specify rules to control what happens when various conditions occur. The following table describes each type of rule.

TABLE 3–4 Rule Types

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 activeSync namespace, as well as the resource ID and name. A process rule controls all functionality that occurs when the system detects any change on the resource. It is used when full control of the account processing is required. As a result, a process rule overrides all other rules. If a process rule is specified, the process will be run for every row regardless of any other settings on this adapter. At minimum, a process rule must perform the following functions: ■ Query for a matching User view. ■ If the User exists, checkout the view. If not, create the User. ■ Update or populate the view. ■ Checkin the User view. It is possible to synchronize objects other than User, such as LDAP Roles.

TABLE 3–4 Rule Types *(Continued)*

Parameter	Description
Correlation Rule	If no Waveset user's resource info is determined to own the resource account, Waveset invokes the Correlation Rule 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). The rule returns one of the following pieces of information that can be used to correlate the entry with an existing Waveset account: ■ Waveset user name ■ WSAttributes object (used for attribute-based search) ■ List of items of type AttributeCondition or WSAttribute (AND-ed attribute-based search) ■ List of items of type String (each item is the Waveset ID or the user name of an Waveset account) If more than one Waveset account can be identified by the correlation rule, you need a confirmation rule or resolve process rule to handle the matches. For the Database Table, Flat File, and PeopleSoft Component Active Sync adapters, the default correlation rule is inherited from the reconciliation policy on the resource. The same correlation rule can be used for reconciliation and Active Sync. See *Correlation and Confirmation Rules* for more information.
Confirmation Rule	Rule that is evaluated for all users that are returned by a correlation rule. For each user, the full User view of the correlation Waveset 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 that can be expressed like a Boolean value. For example, "true" or "1" or "yes" and "false" or "0" or null. For the Database Table, Flat File, and PeopleSoft Component Active Sync adapters, the default confirmation rule is inherited from the reconciliation policy on the resource. The same confirmation rule can be used for reconciliation and Active Sync. See *Correlation and Confirmation Rules* for more information.
Delete Rule	A rule that can expect a map of all values with keys of the form `activeSync.` or `account.` 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 that can be expressed like a Boolean value. For example, "true" or "1" or "yes" and "false" or "0" or null. If the rule returns true for an entry, the account deletion request will be processed through forms and workflow, depending on how the adapter is configured.
Resolve Process Rule	Either the name of the `TaskDefinition` or a rule that returns the name of a `TaskDefinition` to run in case of multiple matches to a record in the feed. The Resolve Process rule gets the resource account attributes as well as the resource ID and name. This rule is also needed if there were no matches and **Create Unmatched Accounts** is not selected. This workflow could be a process that prompts an administrator for manual action.

TABLE 3–4 Rule Types *(Continued)*

Parameter	Description
Create Unmatched Accounts	If set to true, creates an account on the resource when no matching Waveset user is found. If false, Waveset does not create the account unless the process rule is set and the workflow it identifies determines that a new account is warranted. The default is true.
Populate Global	If set to true, populates the global namespace in addition to the activeSync namespace. The default value is false.

If the Adapter Does Not Find the User

If Waveset cannot find a match with an existing Waveset user, it turns an update operation into a create operation if the Create Unmatched Accounts setting is true, or the Resolve Process workflow indicates a `feedOp` of `create`.

The `feedOp` field is available to forms that contain logic to create, delete, or update users. You can use this field to disable or enable fields that are specific to one kind of event (for example, the generation of a password when the `feedOp` field is set to create).

This example `feedOp` field creates a password only when the Active Sync-enabled adapter detects a user on the resource that is not matched by a user in Waveset, and creates the user in Waveset.

EXAMPLE 3–2 Example feedOp Field

```
<Field name='waveset.password'>
   <Disable>
      <neq>
         <ref>feedOp</ref>
         <s>create</s>
      </neq>
   </Disable>
   <expression>
      <cond>
         <notnull>
            <ref>activeSync.password</ref>
         </notnull>
         <ref>activeSync.password</ref>
         <s>change12345</s>
      </cond>
   </expression>
</Field>
```

Using Forms

Active Sync-enabled adapters typically use two types of forms during processing: a *input form* and a *user form*.

Form processing occurs in three steps:

1. Active Sync fields are filled in with attribute and resource information. Use the `activeSync` namespace to retrieve and set attributes on the resource.
2. The input form is expanded and derived. During this expansion, all user view attributes are available.
3. The user form is expanded and derived.

   The `$WSHOME/sample/forms` directory provides sample forms that end with `ActiveSyncForm.xml`. They include logic for handling the cases of new and existing users, as well as logic for disabling or deleting the Waveset user when a deletion is detected on the resource.

Note – Place only resource-specific logic in the input form and include common logic in the user form, possibly enabled when the `feedop` field is not null. If the input form is set to none, all of the Active Sync attributes (except `accountId`) are named global and will propagate automatically.

Input Form

The *input form* is the form that the administrator selects from a pull-down menu when the resource is created or edited. A reference to a selected form is stored in the resource object.

Input forms are used with Active Sync-enabled adapters in the following ways:

- Translate incoming attributes from the schema map.
- Generate fields such as password, role, and organization.
- Provide simple control logic for custom processing, including logic for handling the cases of new and existing users, as well as logic for disabling or deleting the Waveset user when a deletion has been detected.
- Copy and optionally transform attributes from `activeSync` to fields that the user form takes as inputs. The required fields for a creation operation are `waveset.accountId` and `waveset.password`. Other field can be set, too, (for example, `accounts[AD].email` or `waveset.resources`).
- Cancel the processing of the user by setting `IAPI.cancel` to true. This is often used to ignore updates to certain users.

  The following example shows a simple field that will ignore all users with the last name Doe.

EXAMPLE 3–3 Field Ignores All Users with Last Name Doe

```
<Field name='IAPI.cancel'>
   <Disable>
      <neq>
         <ref>activeSync.lastName</ref>
         <s>Doe</s>
      </neq>
   </Disable>
```

EXAMPLE 3–3 Field Ignores All Users with Last Name Doe *(Continued)*

```
   <expression>
      <s>true</s>
   </expression>
</Field>
```

Input forms include logic for handling the cases of new and existing users, as well as logic for disabling or deleting the Waveset user when a deletion has been detected.

User Form

The *user form* is used for editing from the Waveset interface. You assign it by assigning a *proxy administrator* to the adapter. If the proxy administrator has a user form associated with him, this form is applied to the user view at processing time.

Proxy Administrator and the User Form

You set a proxy administrator for an adapter through the `ProxyAdministrator` attribute, which you can set to any Waveset administrator. All Active Sync-enabled adapter operations are performed as though the Proxy Administrator was performing them. If no proxy administrator is assigned, the default user form is specified.

Alternative Form to Process Attributes

Best practice suggests keeping common changes, such as deriving a fullname from the first and last name, in the *user form*. The *input form* should contain resource-specific changes, such as disabling the user when their HR status changes. However, you can alternatively place it in an included form after the desired attributes are placed in a common path, such as incoming.

```
<Form>
   <Field name='incoming.lastname'>
      <ref>activeSync.lastname</ref>
   </Field>
   <Field name='incoming.firstname'>
      <ref>activeSync.firstname</ref>
   </Field>
</Form>
```

Subsequently, in the common form, reference `incoming.`*xxx* for the common logic:

```
<Form>
   <Field name='fullname'>
       <concat>
           <ref>incoming.firstname</ref>
          <s> </s>
          <ref>incoming.lastname</ref>
       </concat>
    </Field>
</Form>
```

Process Cancel Action

To cancel the processing of a user, set `IAPI.cancel` to true in the input form. You can use this to ignore updates to certain users.

Note – If `IAPI.cancel` is set to a value of true in an Active Sync form, then the process associated with an IAPIUser or IAPIProcess event will not be launched.

The following example shows a simple field in the input form that ignores all users with the last name *Doe*.

```
<Field name='IAPI.cancel'>
    <Disable>
      <eq><ref>activeSync.lastName</ref><s>Doe</s></eq>
   </Disable>
    <Expansion>
       <s>true</s>
    </Expansion>
 </Field>
```

Launching Workflow Processes

The Active Sync wizard allows an administrator to specify a pre-poll and post-poll workflow. These workflows are similar in concept to the workflows discussed in *Reconciliation Workflows*.

Some Active Sync-enabled adapters support a resource attribute that runs a specified workflow instead of checking the pulled changes into the user view. This workflow is run with an input variable of only the Active Sync data. For adapters that do not support a separate process, or one where you want to use the standard user form and then launch a process, you can override the process by setting options.

```
<Form>
    <Field name='sourceOptions.Process'>
       <Expansion>
          <s>My workflow process name</s>
       </Expansion>
    </Field>
 </Form>
```

The workflow specified through the form is called just like a standard provisioning workflow. Sun strongly recommends that you base your custom workflow on the standard create and update workflow. Consult the create and update user workflows in `workflow.xml`.

Example: Disabling Accounts through Active Sync-Enabled Adapters

In this example, the resource (an HR database) can be updated with an employee's current status at the company. Based on the input from this HR database, the Active Sync-enabled adapter can disable, delete, create, or perform other actions on the user's accounts across the enterprise by updating the Waveset repository.

The following code example disables all accounts for an employee if there is an incoming attribute called Status and it is not active (“A”). The following table identifies the four states of this attribute.

TABLE 3–5 Attribute States

State	Description
A	active
T	terminated
L	laid off
S	pending change

Based on the value of the Status attribute, the account can be disabled or enabled.

EXAMPLE 3–4 Disabling Accounts for Incoming, Inactive Status Attribute

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Configuration wstype='UserForm' name='PeopleSoft ActiveSync Form'>
   <Extension>
      <Form>
<!-- this is a sample of how to map the accountID to a different field than the
one from the schema map
Commented out because we want to use the default account ID mapped from the resource
Schema Map.
<Field name='waveset.accountId'>
   <Disable>
      <neq>
         <ref>feedOp</ref>
         <s>create</s>
      </neq>
   </Disable>
   <Expansion>
      <concat>
         <s>ps</s>
         <ref>waveset.accountId</ref>
      </concat>
   </Expansion>
</Field> -->

 <!-- this is the real one, limited to create -->
<Field name='waveset.accountId'>
   <Disable>
      <neq>
         <ref>feedOp</ref>
         <s>create</s>
      </neq>
   </Disable>
   <Expansion>
      <ref>activeSync.EMPLID</ref>
   </Expansion>
</Field>
```

EXAMPLE 3–4 Disabling Accounts for Incoming, Inactive Status Attribute *(Continued)*

```
<!-- we need to make up a password for accounts that are being created. This picks
the last six digits of the SSN. -->
<Field name='waveset.password'>
   <Disable>
      <neq>
         <ref>feedOp</ref>
         <s>create</s>
      </neq>
   </Disable>
   <expression>
      <s>change123456</s>
   </expression>
</Field>

<Field name='waveset.resources'>
<!-- <Disable><neq><ref>feedOp</ref><s>create</s></neq></Disable> -->
<!-- Don't change the resources list if it already contains peoplesoft -->
   <Disable>
      <member>
         <ref>activeSync.resourceName</ref>
         <ref>waveset.resources</ref>
      </member>
   </Disable>
   <expression>
      <appendAll>
         <ref>waveset.resources</ref>
         <ref>activeSync.resourceName</ref>
      </appendAll>
   </expression>
</Field>

<!-- Status is mapped by the schema map to PS_JOB.EMPL_STATUS which has at least
four states -
A for active,
T terminated,
L laid off, and
S which is a pending change.
The audit data tells us what the state was, and the global data tells us what
it is. Based on the change we can disable or enable the account Note that this
can happen on a create also! -->

<Field>
   <Disable>
      <eq>
         <ref>activeSync.Status</ref>
         <s>A</s>
      </eq>
   </Disable>
   <Field name='waveset.disabled'>
      <Expansion>
         <s>true</s>
      </Expansion>
   </Field>
   <FieldLoop for='name' in='waveset.accounts[*].name'>
      <Field name='accounts[$(name)].disable'>
```

EXAMPLE 3–4 Disabling Accounts for Incoming, Inactive Status Attribute *(Continued)*

```
         <expression>
            <s>true</s>
         </expression>
      </Field>
   </FieldLoop>
</Field>

<!-- Status is mapped by the schema map to PS_JOB.EMPL_STATUS which has at least
four states -
A for active,
T terminated,
L laid off, and
S which is a pending change.
This is the enable logic. It is disabled if the account status is <> A or is
already enabled -->

<Field>
   <Disable>
      <neq>
         <ref>activeSync.Status</ref>
         <s>A</s>
      </neq>
   </Disable>
   <Field name='waveset.disabled'>
      <Disable>
         <eq>
            <ref>waveset.disabled</ref>
            <s>false</s>
         </eq>
      </Disable>
      <Expansion>
         <s>false</s>
      </Expansion>
   </Field>
   <FieldLoop for='name' in='waveset.accounts[*].name'>
      <Field name='accounts[$(name)].disable'>
         <Expansion>
            <s>false</s>
         </Expansion>
      </Field>
   </FieldLoop>
</Field>
</Form>
</Extension>
<MemberObjectGroups>
<ObjectRef type='ObjectGroup' id='#ID#Top' name='Top'/>
</MemberObjectGroups>
</Configuration>
```

CHAPTER 4

Dataloading Scenario

This chapter provides tips to consider when preparing to load account information into Waveset. It also includes sample scenarios that illustrate some issues that you might encounter.

Assessing Your Environment

Before you can begin loading user account information into Waveset, you determine which know the following questions applies to your environment:

- Is there an authoritative resource for all user IDs?

  If yes, then loading user accounts into Waveset should be straightforward. Use that resource as your first resource, then load accounts from other resources, using correlation rules to link the accounts together.
- Can a complete list of users be obtained from resources that overlap, but for which there is a correlation key?

  If yes, then the process of loading user accounts will be similar. Be sure that the user accounts are loaded from the overlapping resources into Waveset before loading accounts from other resources.
- Can a list of users be obtained from overlapping resources that do not have a correlation key?

  If yes, then determine how a unique set of users can be discovered from those resources most easily. You will probably need to manually correlate and delete users for each resource.

If the answer is no to all these questions, then the process of loading accounts is problematic. Load user accounts as best you can, and plan to delegate creation of other Waveset users to departmental administrators or end-users.

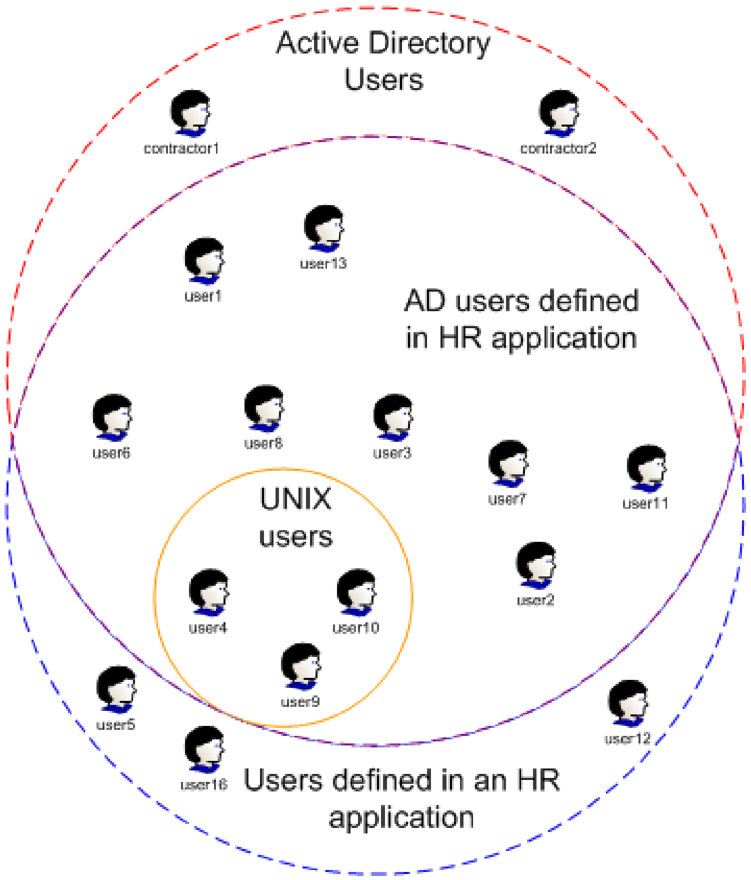
Choosing the First Resource

Ideally, the first resource you use to load accounts into Waveset has the following characteristics:

- **References a comprehensive set of users**. The goal of an initial load is get as many accounts into Waveset as possible. Thus, the following applications might be a good choice:
  - A Human Resources application, such as PeopleSoft or SAP. (If the application does not contain contractors and other temporary workers, be sure to load those accounts separately.)
  - A directory-based application, such as LDAP or Active Directory. A majority of users are often defined in a central organization or organization unit.
- **Contains enough information to construct an Waveset account ID**. Each Waveset account ID must be unique. Ideally, your resource will have an attribute that is guaranteed to be unique and can be used as a Waveset account ID. Examples include an employee IDs or Active Directory sAMAccountName attributes. First and last names can also be concatenated to produce an account ID, but this technique might not guarantee a unique Waveset account will be generated.
- **Stores user attributes that can be used correlation keys.** To link resource accounts in Waveset, you must have attributes that have the same values across two or more resources. Ideally, the values on the secondary resources will always perfectly match values on the first resource. In addition, it would be ideal if the values on the secondary resource are unique within that resource. The best attributes include employee ID and full name, but any other attribute that is present and consistent across multiple resources is acceptable.
- **Contains data that can be considered authoritative.** If users can edit their own account data, then the data might not be consistent across systems.

The following diagram illustrates a small scenario in which a company has three types of resources. Most of the company's workers are defined in a Human Resources application, such as PeopleSoft or SAP. However, the company does not enter contractors in the HR application, so the contractors cannot be loaded into Waveset using this application. The Active Directory also defines most, but not all, users. (These users might be factory workers with no need for computer access.) Thus, the majority of users are defined in both resources, but neither contains all the users. Some workers also have UNIX accounts.

FIGURE 4–1 Small Dataloading Scenario



Which resource should be selected as the first resource? The UNIX resource can be safely eliminated, because it does not contain a comprehensive set of users. Active Directory and the HR application contain about the same number of users, so neither has a clear advantage.

Factors that can help determine whether the Active Directory or HR application should be loaded first include the following:

- Urgency to manage accounts within Waveset. If the workers not defined in Active Directory (that is, they are defined in the HR application only) do not have any additional resource accounts, such as UNIX or other systems, then the HR application might not be as crucial as the Active Directory resource.
- Correlation keys. If one resource has attributes that are also on the UNIX accounts, then that attribute might be a better choice.

- Waveset login names. If one resource creates a more desirable login name of Waveset, then this can be a deciding factor.

Choosing the First Data Loading Process

After you have chosen which resource will be used as the starting point for loading user data into Waveset, you must decide which process to use. The following table provides a summary of the benefits and drawbacks of each data loading process. A discussion of each data loading process follows.

TABLE 4–1 Overview of Data Loading Processes

Data Loading Process	Advantages	Disadvantages
Load from File	■ Quickest loading process. ■ Easy to control which attributes are loaded. ■ Easier to configure and faster than reconciliation.	■ Requires customer time to generate a CSV file from a resource. ■ Requires a full reconciliation before production. ■ Cannot be used to update accounts.
Load from Resource	■ Works with all resources. ■ Easier to configure than reconciliation.	■ Cannot pick and choose which resource accounts will be loaded. ■ Requires a full reconciliation before production.
Create bulk action	Allows you to add multiple accounts simultaneously to an Waveset user.	■ Slower than loading from resource or reconciliation. ■ Cannot easily generate the CSV file from resources. ■ Requires detailed knowledge of Waveset to make full use of this feature. ■ Requires a full reconciliation before production.
Reconciliation	■ Can implement all aspects of reconciliation policy. ■ Using reconciliation up-front prevents last-minute surprises.	■ Cannot pick and choose which resource accounts will be loaded. ■ Can take a large amount of time to load all accounts in large environments (over 50,000 employees).
ActiveSync	If at all possible, avoid choosing ActiveSync as the means to load account information. ActiveSync is designed to detect changes, and as a result, initial loads are slow.	

Using Load from File

The Load from File process seeds Waveset accounts with basic values, such as account ID, first and last name, and e-mail address. The account ID is the only required attribute.

The Load from File process imports the contents of a comma-separated values (CSV) file into Waveset. The top line of this file contains a list of attribute names, separated by commas. Each subsequent line contains a series of corresponding attribute values. All attributes must also be separated by commas.

Note – .Load from File also accepts XML files, but the syntax of an XML file must match the syntax generated by the Extract to File feature. This format is beyond the scope of this discussion.

Load from File also accepts XML files, but the syntax of an XML file must match the syntax generated by the Extract to File feature. This format is beyond the scope of this discussion.

The data in a CSV file is often exported from a resource. For example, the Active Directory Users and Computers MMC (Microsoft Management Console) allows you to export the contents of an organization unit directly into a CSV file. The console exports all users defined in the organization unit as well as the displayed attributes. Therefore, you should verify only attributes that will be managed by Waveset are displayed in the Active Directory Users and Computers MMC. Including extraneous attributes will cause loading times to increase.

Some resources are not capable of directly exporting user account information to CSV format. If you wish to use to this method of data loading, you might need to extract the information programmatically or add data manually.

For example, the first three lines of a CSV file might look like this:

```
accountId,firstname,lastname,EmployeeID
Josie.Smith,Josie,Smith,1436
AJ.Harris,Anthony,Harris,c310
```

The attributes listed in the CSV file must be pre-defined as user view attributes. Basic attributes such as `accountId`, `email`, `password`, and `confirmpassword` are pre-defined. Others are defined in the extended user attributes configuration object. By default, this object adds `firstname`, `lastname`, and `fullname` to the list of available attributes.

If you want to retain values for attributes that are not pre-defined in Waveset, such as an employee ID, you must add them to the Extended User Attributes Configuration object.

The Load from File process configuration page prompts for a correlation and confirmation rules. Since this is the first attempt to load data, select the *User Name Matches AccountId* correlation rule. You do not need a confirmation rule.

It is important to remember that the data contained in the CSV file is used for Waveset accounts only. Even if the data is exported directly from Active Directory, for example, the data is not linked to any Active Directory accounts or resources unless you have created a custom user form to do this. Without a custom user form, a different data loading mechanism must be used to link resource account data to an Waveset user. User forms are discussed briefly in "Assigning User Forms" on page 69.

Note – .The Load from File process does not add entries into the Waveset account index. Therefore, you must perform a full reconciliation, or update the users, before your Waveset deployment is complete. In addition, the Load from File does not run any workflows when creating users in Waveset.

The Load from File process does not add entries into the Waveset account index. Therefore, you must perform a full reconciliation, or update the users, before your Waveset deployment is complete. In addition, the Load from File does not run any workflows when creating users in Waveset.

Using Load from Resource

Although the configuration pages for the Load from Resource and Load from File processes are almost identical, the Load from Resource is functionally closer to reconciliation. The Load from Resource and reconciliation processes pull data from the resource, and then adds the accounts it finds to Waveset. Therefore, the adapter must be configured before you perform either of these operations.

Load from Resource is faster on the initial run than reconcile but does not populate the Account Index. Reconcile on the second execution running in Incremental mode should be faster than Load from Resource. If reconciliation is the desired tool as the long-term solution, then use reconciliation for the initial load of users.

A user form can be used to set the account ID, place users in an Organizations, and perform other related tasks related to creating users. See "Assigning User Forms" on page 69 for more information.

When you seed Waveset accounts for the first resource using Load from Resource, the correlation and confirmation rules are not very meaningful. Select the **User Name Matches AccountId** correlations rule. You do not need a confirmation rule.

Note – .The Load from Resource process does not add entries into the Waveset account index. Therefore, you must perform a full reconciliation, or update the users, before your Waveset deployment is complete. In addition, the Load from File does not run any workflows when creating users in Waveset.

Using Create Bulk Actions

The Create bulk action loads data from a CSV file. Unlike the Load from File process, the create bulk action allows you to define any writable attribute in the user view, including Waveset-specific attributes, global attributes, and resource account attributes. This flexibility means that it will probably be more difficult to assemble a bulk actions CSV file. If your bulk actions affect multiple resources, you will need to find a way to merge resource data into a single CSV file. However, you could also define a simpler bulk action file and use subsequent update actions to load data into Waveset user accounts.

Bulk actions runs the default workflows for create, update, and delete actions. This slows down the process of loading user accounts, but add greater flexibility.

The following example illustrates the use of Create bulk actions only. The command and user attributes are required.

```
command,user,waveset.resources,password.password,password.confirmPassword,
accounts[MyAD].description,accounts[MySolaris].comment

Create,John Doe,MyAD|MySolaris,changeit,changeit,John Doe,John Doe
Create,Jane Smith,MyAD,changeit,changeit,Jane Smith
```

The following example illustrates how the Create and Update bulk actions can be used in two separate files.

```
command,user,waveset.resources,password.password,password.confirmPassword,
accounts[MyAD].description,
Create,John Doe,MyAD,changeit,changeit,John Doe,John Doe
Create,Jane Smith,MyAD,changeit,changeit,
Jane Smith

command,user,waveset.resources,password.password,password.confirmPassword,
accounts[MySolaris].comment
Update,John Doe,MySolaris,changeit,changeit,John Doe
Update,Jane Smith,MySolaris,changeit,changeit,Jane Smith
```

Creating accounts using bulk actions does not add entries into the Waveset account index. Therefore, you must perform a full reconciliation before your Waveset deployment is complete.

Note – . Creating accounts using bulk actions does not add entries into the Waveset account index. Therefore, you must perform a full reconciliation before your Waveset deployment is complete.

Using Reconciliation

The first reconciliation of a resource will probably take longer than any subsequent reconciliation. You can expect the first reconciliation of a resource to add a large number of Account Index entries.

Preparing for Data Loading

Review the following sections before you begin the process of loading account information into Waveset:

- Configuring an Adapter
- Setting Account ID and Password Policies
- Creating a Data Loading Account
- Assigning Forms

Configuring an Adapter

To manage accounts on resources, you must configure an adapter for each source of account information. If you are using the Load from File process or bulk actions, then the adapter configuration can wait until you are ready to reconcile. Otherwise, the adapter must be configured before you can load data into Waveset.

For general information about configuring an adapter, see "Understanding and Managing Identity Manager Resources" in *Sun Identity Manager 8.1 Business Administrator's Guide*. For detailed information about a specific adapter, refer to the *Sun Identity Manager 8.1 Resources Reference* or the online help.

Setting Account ID and Password Policies

When you load account data from a resource using Load from Resource, reconciliation, or Active Sync, Waveset does not obtain the password from the resource. (It would be a security breach on the part of the resource if it yielded the password.) Therefore, the Waveset account passwords will not be the same as the those on the resource. By default, Waveset generates a random password that must be reset. However, you can also use the password view in the user form to specify a temporary password, such as a literal string that is the same for everyone, or is the same as the Waveset account ID. See Assigning User Forms and Chapter 3, "Identity Manager Views," in *Sun Identity Manager Deployment Reference* for more information.

For bulk actions, and Load from File, you can specify password values in the CSV file. These should be considered temporary passwords that users must change.

Policies establish limitations for Waveset accounts, and are categorized as:

- Waveset account policies -- Use these to establish user, password, and authentication policy options. These account policies are assigned to organizations or users.
- Resource password and account ID policies -- Use these to set or select length rules, character type rules, and allowed words and attribute values.

Make sure you make any updates to the default policies before you begin loading account information into Waveset.

The following table lists the policies provided with Waveset as well as the default settings.

TABLE 4–2 Default Waveset Policies

Policy Name	Default Characteristics
AccountId Policy	Account IDs must have a minimum length of 4 characters and a maximum length of 16 characters.
Default Lighthouse Account Policy	Sets the account ID and password policies to **AccountId Policy**, and **Password Policy**. Passwords are generated by Waveset, rather than by users.
Password Policy	Passwords must have a minimum length of 4 characters and a maximum length of 16 characters. The password cannot contain the user's e-mail, first name, last name, or full name.
Windows 2000 Password Policy	Passwords must have a minimum length of 6 characters. Passwords must have 3 of the following characteristics: ■ 1 numeric character ■ 1 uppercase letter ■ 1 lowercase letter ■ 1 special character In addition, the password cannot contain the account ID.

See Chapter 3, "User and Account Management," in *Sun Identity Manager 8.1 Business Administrator's Guide* for more information about account and password policies.

Creating a Data Loading Account

It is recommended that you create a separate administrator account to perform data loading for the following reasons:

- Data loading actions can be tracked more easily when auditing is enabled.
- Every Waveset administrator is assigned a user form that creates and edits Waveset users. When you create an account for data loading, you can specify a streamlined form that runs quicker than the default forms. See the next section for information.

See "Creating Users and Working with User Accounts" in *Sun Identity Manager 8.1 Business Administrator's Guide* for more information about creating accounts.

Assigning User Forms

In the context of data loading, user forms are used to perform background processing. For example, forms can work in conjunction with resource adapters to process information from an external resource before storing it in the Waveset repository. They can also be used to place users in the correct Organization based on input user data.

The user view is a data structure that contains all available information about an Waveset user. It includes:

- Attributes stored in the Waveset repository
- Attributes fetched from resource accounts
- Information derived from other sources such as resources, roles, and organizations

Views contain many attributes, and a view attribute is a named value within the view (for example, `waveset.accountId` is the attribute in the user view whose value is the Waveset account name).

Most form field names are associated with a view attribute. You associate a field with a view attribute by specifying the name of the view attribute as the name of the form field. For more information on the user view, including a reference for all attributes in the user view, see the chapter titled *Views*.

The following fields are often in a user form that loads users.

- `firstname`
- `lastname`
- `fullname`
- `email`
- `waveset.accountId`
- `waveset.organization`
- `EmployeeId`

The `waveset.accountId` and `waveset.organization` are values specific to Waveset. The `EmployeeId` attribute is a customized attribute. Its use is illustrated in *Defining Custom Correlation Keys*.

Waveset provides numerous forms that are pre-loaded into the system. Additional forms are also available in the `$WSHOME/sample/forms` directory. Many of the forms in this directory are resource-specific. You might wish to review these forms with the Identity Manager IDE to determine whether they should be used in production.

To increase performance during bulk operations, the user form assigned to an administrator should be as simple as possible. If you want to create a form for data loading, then you can remove code that is designed to display data. Another example of simplifying the form would be if you use bulk add actions. Your CSV file could define basic attributes such as `firstname` and `lastname`. These attributes could then be removed from the administrator's user form. See the chapter titled *Waveset Forms* for more information about creating and editing forms.

Note – Do not directly modify a form provided with Waveset. Instead, you should make a copy of the form, give it a unique name, and edit the renamed copy. This will prevent your customized copy from being overwritten during upgrades and service pack updates.

Linking to Accounts on Other Resources

Waveset uses correlation and confirmation rules to link accounts. A correlation rule looks for Waveset users that might own an account. It returns a list of users that match the criteria defined in the correlation rule. A confirmation rule tests an Waveset user against an account to determine whether the user actually does own the account. It returns true or false values. This two-stage approach allows Waveset to optimize correlation, by quickly finding possible owners (based on name or other attributes), and by performing expensive checks only on the possible owners.

Before you begin using correlation and confirmation rules, you must be familiar with the data that is present from the first data load. The Waveset `accountId` will always be present. If you performed a Load from File or a Create bulk action, then the values in the heading row of the CSV file are also present. If you performed a Load from Resource or reconciliation, some key attributes found on the resource will be present, but others will be present only if they are explicitly saved.

In addition, you must be familiar with the account data stored on the secondary resources as well. Ideally, a secondary resource contains data that overlaps with data that has already present in Waveset.

This can be more difficult than it sounds. Different resources often have varying requirements for user accounts. As an example, the following table compares the requirements and restrictions for a Windows account name and a Solaris account name.

TABLE 4–3 Comparison of Windows Account Name and Solaris Account Name Requirements

Characteristics	Windows	Solaris
Maximum length	20 characters	8 characters
Special characters permitted	All but " / \ [ ] : ; | = , + * ? < >	period (.), underscore (_), and hyphen (-) only
Able to specify a full name	Yes	There is one comment field. Traditionally, the comment field lists the user's full name, but other information could be included.
Able to specify additional comments, such as employee ID	Yes	

The differences between Windows and Solaris account names highlight some of the difficulties in linking accounts:

- Because of the differences in maximum length, the account names will usually be different on these two systems. On Windows, users often have an account name that matches their first and last name. On Solaris, account names might be the concatenation of the first letter of the first name plus the first seven digits of the last name. Therefore, a correlation rule that compares account IDs will probably not be enough to link a Solaris account to a Windows account.
- To create a user account on Windows, the administrator must supply a display name. Solaris user accounts do not require display names, although the optional GECOS field can be used to specify a user's name. There are no guidelines about the contents or format of this field. A user's GECOS field might be blank or contain text unrelated to the user's Windows display name. Therefore, a correlation rule that compares full names might not trigger as often as you would expect.

Consider the following questions as you prepare to link accounts:

- Do you have users with similar names, such as Mary A. Jones and Mary B. Jones, or John Doe Jr. and John Doe III? If so, how will you distinguish between them?
- Were account names on each resource defined in a consistent manner? If yes, then it will be easier to define a rule that compares an account ID on a resource with a value stored in Waveset.
- Did users have the ability to edit resource account data? If yes, then the data might not match values that are stored on a system that users cannot change.

Defining Custom Correlation Keys

A rule cannot compare an account value on a resource with an Waveset value unless the value is stored in the system. The `accounts[Lighthouse]` attribute stores many of these values, but additional values must be added with the Extended User Attributes Configuration object. The system does not save attributes that are not registered in the configuration object.

By default, the following attributes are included as extended user attributes:

- `firstname`
- `lastname`
- `fullname`

Note – The `fullname` extended user attribute must be added to the list of QueryableAttrNames.

If you want to use a different attribute, such as an employee ID as part of a correlation rule, then you must add it to the User Extended Attributes configuration object. Use the following steps to do this:

▼ To Define a Custom Correlation Key

1. **Access the Waveset debug page at `http://PathToIDM/debug`. The System Settings page is displayed.**
2. **Select Configuration from the List Objects pull-down menu. The List Objects of type: Configuration page is displayed.**
3. **Select the edit link for User Extended Attributes.**
4. **Add the new attributes to the List element, for example:**

   ```
   <String>EmployeeId</String>
   ```
5. **The attribute must be defined as an Waveset attribute on the Account Attributes (schema map) page for the resource.**
6. **Save your changes. Waveset returns to the System Settings debug page.**

   The custom attribute must also be added to the `QueryableAttrNames` element in the `UserUIConfig` configuration object.
7. **Select Configuration from the List Objects pull-down menu. The List Objects of type: Configuration page is displayed.**
8. **Select the edit link for UserUIConfig.**
9. **Add the new attributes to the `<QueryableAttrNames><List>` element, for example:**

   ```
   <String>EmployeeId</String>
   ```
10. **Save your changes. Waveset returns to the System Settings debug page.**
11. **Restart your application server.**

Creating Custom Rules

Waveset predefines a number of correlation and confirmation rules in `sample/reconRules.xml`. You can use these as a basis for your own rules. Rules must be assigned a subtype of `SUBTYPE_ACCOUNT_CORRELATION_RULE` or `SUBTYPE_ACCOUNT_CONFIRMATION_RULE`.

The following rule compares the `account.EmployeeId` attribute, which is defined on the secondary resource, with the `EmployeeId` attribute that was previously loaded into Waveset. If the secondary resource has an `account.EmployeeId` value, then the correlation rule returns a list of users that match the `EmployeeId`.

```
<Rule subtype='SUBTYPE_ACCOUNT_CORRELATION_RULE' name='Correlate Employee IDs'
   <cond>
      <ref>account.EmployeeId</ref>
      <list>
         <new class='com.waveset.object.AttributeCondition'>
            <s>EmployeeId</s>
            <s>equals</s>
            <ref>account.EmployeeId</ref>
         </new>
      </list>
   </cond>
</Rule>
```

In this example, the `EmployeeId` attribute has been previously added to the User Extended Attributes and `UserUIConfig` configuration objects. If this attribute has was not included as a default Waveset attribute name for the resource, it must also be added or edited on the schema map for the resource.

Correlation rules return a list of possible matches. If the results are expected to return only one match, such as an employee ID, then no confirmation rule would be needed. However, if there could be multiple matches, which could be the case if correlation found accounts that matched by first and last name, then a confirmation rule would be needed to further identify the match.

Rules can be added to Waveset by using the Identity Manager IDE, importing an XML file, or editing and renaming an existing rule using the debug page.

Manually Linking Accounts

Waveset provides several mechanisms that can be used to assign accounts when correlation and confirmation rules do not find a match.

Using the Account Index

The Account Index records the last known state of each resource account known to Waveset. It is primarily maintained by reconciliation, but other Waveset functions will also update the Account Index, as needed.

Note – Load from resource, load from file, and bulk actions do not update the Account Index.

To view the account index, click the Resources tab, then click the Account Index link on the left. Then navigate to a resource to display the status of all accounts on that resource.

When you right-click on an uncorrelated account (represented in the Account Index table with a situation of "UNMATCHED" and an Owner of "_UNKNOWN_"), Waveset displays a menu that presents you with the options of creating a new Waveset user account, running reconciliation on a single account using the reconciliation policy in effect for the resource, specifying an owner, or deleting or disabling the resource account. If you select the “Specify Owner” option, Waveset displays a screen that allows you to search for owners that might criteria that you specify. Refer to Business Administrator's Guide for more information.

Enabling Self-Discovery

The Waveset User Interface can be configured to allow Waveset users to discover their own resource accounts. This means that a user with an Waveset identity can associate it with an existing, but unassociated, resource account. Self-discovery can be enabled only on resources that support pass-through authentication.

To enable self-discovery, you must edit the End User Resources configuration object, and add to it the name of each resource on which the user will be allowed to discover accounts.

▼ To Enable Self-Discovery

1 **Access the Waveset debug page at `http://PathToIDM/debug`. The System Settings page is displayed.**

2 **Select Configuration from the List Objects pull-down menu. The List Objects of type: Configuration page is displayed.**

3 **Select the edit link for End User Resources.**

4 **After the `<List>` element, add `<String>Resource</String>`, where `Resource` matches the name of a resource object in the repository. For example, to allow users to self-discover their accounts on resources `AD` and `Solaris`, edit the `<List>` element as follows:**

```
<List>
   <String>AD</String>
   <String>Solaris</String>
</List>
```

5 **Save your changes. Waveset returns to the System Settings debug page.**

When self-discovery is enabled, the user is presented with a new menu item on the Waveset User Interface (**Inform Waveset of Other Accounts**) This area allows him to select a resource from an available list, and then enter the resource account ID and password to link the account with his Waveset identity.

Example Scenarios

This section provides scenarios that illustrate the process of loading accounts from one or more resources. The following scenarios discuss issues that might arise in your environment.

- Active Directory, SecurID, and Solaris
- LDAP, PeopleSoft, and Remedy
- Expedited Bulk Add

Active Directory, SecurID, and Solaris

A company wants to use Waveset to manage Active Directory, SecurID, and Solaris accounts. All workers have an Active Directory account, and most employees have a SecurID account. Only a fraction of employees have a Solaris account. After examining the account data on each resource, the Waveset administrator has determined the following attributes can be used as correlation keys.

TABLE 4–4 Possible Correlation Keys

Possible Correlation Keys	Active Directory	SecurID	Solaris
Account ID matches AD	N/A	Yes	No
Employee ID	Yes	No	No
Full name	Yes	Yes	Yes (Description attribute)

Because all employees have an Active Directory account, it will be used as the first data loading resource. SecurID will be loaded second, because the account IDs on this resource match those on Active Directory. Account IDs are always unique, therefore this is a better correlation key than full name. The Active Directory and SecurID accounts are expected to correlate without problems.

Correlating the Solaris accounts will be difficult. The only correlation attribute that exists on Solaris accounts is the user's full name. Solaris does not have individual attributes for defining first name and last name. As a result, the correlation rule will be a comparison of the string defined in the Solaris `useradd -c` command with the `fullname` value in Active Directory. The comparison will often fail, due to factors such as use of nicknames or extraneous spaces and punctuation.

Example Users

In this scenario, the following users demonstrate some of the possible problems you might encounter when loading accounts.

TABLE 4–5 Dataloading Scenario: Potential Problems during Account Loading

Worker name	AD and SecurID Logon Name	AD Full Name	Solaris Account Name	Solaris Description
Anthony Harris	AJ Harris	Anthony J Harris	ajharris	A.J. Harris
Isabelle Moreno	Isabelle Moreno	Isabelle Moreno	imoreno	Isabelle Moreno
John Thomas (Sr.)	John Thomas	John Thomas	jthomas	John Thomas
John Thomas (Jr.)	John P. Thomas	John P. Thomas	jthomas2	John Thomas
Robert Blinn	Robert Blinn	Bob Blinn	rblinn	Bob Blinn
Theodore Benjamin	Theodore Benjamin	Theodore Benjamin	tbenjami	Ted Benjamin

Loading Active Directory Accounts

Use the following steps as a guideline for using reconciliation to load Active Directory accounts into Waveset.

▼ To Load Active Directory Users

1. **From the Resources page in the Administrator Interface, select the Windows 2000/ Active Directory resource from the New Resource pull-down menu. Then configure the adapter.**

   Make sure you do not delete the `accountId` or `fullname` Waveset user attribute from schema map. Also make sure the identity template is correct. See the online help and the Resource Reference for more information about configuring the adapter.

2. **(Optional) Edit the account and password policies as desired. See** *Setting Account ID and Password Policies* **for more information.**

3. **(Optional) Create a user form that will be used for reconciliation. See** *Assigning User Forms* **for more information.**

4. **(Optional) Create an Waveset user for performing data loading. Assign the user form created in the previous step to the user.**

5. **Configure the reconciliation policy for the resource. On the first resource, the correlation rule is not important, and the confirmation rule is not used when creating Waveset users. Since this is the first resource, you probably want to assign the UNMATCHED situation to the value "Create new Waveset user based on resource account."**

6. **If you created a user to perform data loading, log in as that user. This step is not necessary for reconciliation, but would be for Load from File, Load from Resource, or Bulk actions.**

7. **Reconcile the Active Directory resource.**

Results

If you used the default Waveset account policy and default Active Directory identity template, Waveset will not create an Waveset user that links to Theodore Benjamin's Active Directory account, because his name contains more than 16 characters. For this example, the account ID policy was set to 25 characters.

Waveset creates user accounts for all resource accounts with a situation status of CONFIRMED. This should include all users that passed the password and account ID policies. Unless your user form specified otherwise, the Waveset account name will be the same as Active Directory login name.

Loading SecurID Accounts

When SecurID is implemented, SecurID user records are usually imported from a Microsoft Security Accounts Manager (SAM) database or from an LDAP server. As a result, the SecurID account IDs match those from the source. This makes correlating users a relatively simple task, because there is a one-to-one correlation between SecurID and Active Directory accounts. The User Name Matches Account ID correlation rule can be used to quickly link these accounts.

To load SecurID accounts, perform the procedure described in "Loading Active Directory Accounts" on page 77, with the following modifications:

- When you are configuring the SecurID adapter, ensure that you do not delete the `accountId` Waveset user attribute.
- Configure the reconciliation policy as follows:
  - Set the correlation rule to "User Name Matches Account ID."
  - Since Active Directory is considered to be an authoritative source, and SecurID relies on Active Directory account information, you might want to set the UNMATCHED situation option to "Delete Resource Account" or "Disable Resource Account." The UNASSIGNED situation should be set to "Link resource account to Identity Manager user."

All SecurID accounts should correlate with the Active Directory account. Perform any additional steps to resolve UNMATCHED or DISPUTED situations.

Loading Solaris Accounts

In this scenario, the `fullname` attribute is the only correlation key. This is a weak correlation key, because differences in spacing and punctuation guarantee matches will fail. In addition, users can change their display names with the Solaris `chfn` command. Even if full names once matched, they might not agree if any users have run the `chfn` command.

By default, the `fullname` attribute is not queryable. To enable this feature, you must edit the `UserUIConfig` configuration object, and add the `fullname` attribute to the `<QueryableAttrNames><List>` element. See *Defining Custom Correlation Keys* for more information.

You will also need to create a custom rule to correlate `fullname` attributes. The following example, which is named “Correlate Full Names”, performs the correlation. It compares the value of the `account.Description` attribute from the Solaris resource to the `fullname` attribute, a system attribute that was populated from Active Directory.

```
<Rule subtype='SUBTYPE_ACCOUNT_CORRELATION_RULE' name='Correlate Full Names'
   <cond>
      <ref>account.Description</ref>
      <list>
         <new class='com.waveset.object.AttributeCondition'>
            <s>fullname</s>
            <s>equals</s>
            <ref>account.Description</ref>
         </new>
      </list>
   </cond>
</Rule>
```

This rule compares the Description attribute from the Solaris resource with the Waveset `fullname` attribute. If the two attributes match, the accounts are correlated, with a situation of CONFIRMED.

To load Solaris accounts, perform the procedure described in “Loading Active Directory Accounts” on page 77, with the following modifications:

- When you are configuring the Solaris adapter, ensure that you do not delete the `accountId` or Description Waveset user attribute.
- Configure the reconciliation policy as follows:
  - Set the correlation rule to “Correlate Full Names” (the example rule).
  - There could be numerous Solaris accounts that do not correlate with the accounts already loaded into Waveset. Set the UNASSIGNED situation to “Link resource account to Identity Manager user”. In most cases, you should set the UNMATCHED situation to “Do nothing”. Deleting or disabling unmatched users could result with a loss of data or productivity.

The following table describes the users in this dataloading scenario.

TABLE 4–6 Users in Dataloading Scenario

Worker name	AD Full Name	Solaris Account Name	Solaris Description
Anthony Harris	Anthony J Harris	ajharris	A.J. Harris
Isabelle Moreno	Isabelle Moreno	imoreno	Isabelle Moreno
John Thomas (Sr.)	John Thomas	jthoma	John Thomas
John Thomas (Jr.)	John P. Thomas	jthomas2	John Thomas
Robert Blinn	Bob Blinn	rblinn	Bob Blinn

TABLE 4–6 Users in Dataloading Scenario *(Continued)*

Worker name	AD Full Name	Solaris Account Name	Solaris Description
Theodore Benjamin	Theodore Benjamin	tbenjami	Ted Benjamin

In this example, only accounts for Isabelle Moreno can be expected to correlate.

- The accounts for Anthony Harris, John Thomas (Jr.), Robert Blinn, and Theodore Benjamin will not correlate because the Active Directory `fullname` attributes do not exactly match the Solaris Description attributes. These accounts will have a situation of UNMATCHED. In this scenario, the Solaris account names are based on first initial plus last name. With the exception of the John Thomas account, assigning these unmatched Solaris accounts is easy.
- The Solaris accounts jthomas and jthomas2 will have a situation of DISPUTED. Both of these accounts have a Description value of John Thomas. You must find another means to determine which user Solaris accounts jthomas and jthomas2 belong to. Ideally, you could use a confirmation rule to distinguish between the two accounts. However, Solaris and Active Directory accounts do not contain enough intersecting attributes to create a confirmation rule.

LDAP, PeopleSoft, and Remedy

In this scenario, the LDAP or PeopleSoft resource could theoretically be the primary resource.

- If all employees and contractors are tracked in PeopleSoft, then this application can be considered an authoritative resource.
- If all employees have LDAP accounts, then LDAP could be considered an authoritative resource.

Remedy is not a candidate to be the primary resource, because only a small percentage of workers have a Remedy account.

In many cases, if you have multiple authoritative resources, then any of those resources can be loaded first. However, the PeopleSoft Component adapter performs Active Sync functions only. (There is another PeopleSoft adapter available, but it is limited in scope.) The PeopleSoft Component adapter does not perform reconciliation, and as a result, reconciliation policy cannot be set for the resource. There are no correlation rules available, so the PeopleSoft accounts must be loaded first. The Waveset account names will match PeopleSoft EMPLID (employee ID) values.

The PeopleSoft employee ID is ideal as a correlation key, because it is unique for all users defined in the system. The LDAP inetOrgPerson object contains an employeeNumber attribute. An employee ID could also be stored in an attribute with a label such as Description, or in a custom attribute. This scenario assumes the employeeNumber LDAP attribute is in use.

The Remedy adapter does not provide default attributes. You must customize the adapter to fit your environment. Because the Remedy application is often configured to send email when a request enters the system, the e-mail attribute should be available. The LDAP inetOrgPerson object also contains the mail attribute. Therefore, the e-mail address will be the correlation key.

The following table lists the correlation keys for each resource in this scenario.

TABLE 4–7 Dataloading Scenario: Correlation Keys for Each Resource

Possible Correlation Keys	PeopleSoft	LDAP	Remedy
Employee ID	Yes	Yes	No
E-mail address	No	Yes	Yes

Example Users

In this scenario, the following users demonstrate some of the possible problems you might encounter when loading accounts.

TABLE 4–8 Deployment Scenario: Possible Problems during Account Loading

Worker name	PeopleSoft EMPLI	LDAP EmployeeNumber	LDAP Email(@example.com)	Remedy Email(@example.com)
Robert Blinn	945	945	Bob.Blinn	bblinn
William Cady	None	None	William.Cady	William.Cady
Eric D'Angelo	1096	1096	Eric.D'Angelo	Eric.D'Angelo
Renée LeBec	891	None	None	None
Josie Smith	1436	1463	Josie.Smith	None
John Thomas	509	509	John.Thomas	None
John P. Thomas	None	None	John.P.Thomas	John.P.Thomas

Loading PeopleSoft Users

Use the following steps as a guideline for using reconciliation to load PeopleSoft accounts using Active Sync into Waveset.

▼ To Load PeopleSoft Users

1 **From the Resources page in the Waveset Administrator Interface, select the PeopleSoft Component resource from the New Resource pull-down menu. If this resource is not displayed, click the Configure Managed Resources button and add**

`com.waveset.adapter.PeopleSoftComponentActiveSyncAdapter` as a custom resource. This adapter requires the installation of a JAR file provided by PeopleSoft. See the Resource Reference for more information.

2 **Configure the adapter. Make sure you do not delete the `accountId` or `fullname` Waveset user attribute from schema map. Also make sure the identity template is correct.**

3 **(Optional) Edit the account and password policies as desired. See** *Setting Account ID and Password Policies* **for more information.**

4 **(Optional) Create a user form that will be used for data loading. The `$WSHOME/sample/forms/PeopleSoftForm.xml` file can be used as a foundation. See** *Assigning User Forms* **for more information.**

5 **Start ActiveSync on the PeopleSoft adapter.**

Results of This Scenario

Waveset loads all users unless the user form indicates that an account should not be loaded. In this scenario, Renée LeBec does not have an LDAP or Remedy account. Presumably, she no longer works for the company. If you used the default PeopleSoft form, then Waveset disables PeopleSoft accounts for terminated employees.

The accounts for William Cady and John P. Thomas are not created because they are not defined within PeopleSoft.

Loading LDAP Users

In this scenario, the `employeeNumber` attribute in the LDAP `inetOrgPerson` object is the correlation key. This attribute is not listed by default in the schema map for the LDAP adapter, so you must add it manually. For this example, add the attribute `EmployeeId` to the Waveset User Attribute side of the schema map, and `employeeNumber` to the Resource User Attribute side.

Note – The PeopleSoft adapter uses the Waveset attribute name `EmployeeId` by default. This value was chosen to maintain consistency between LDAP and PeopleSoft, although this is not required.

The e-mail address will be the correlation key for the Remedy resource, but it must be set-up and configured before you load LDAP accounts. The `inetOrgPerson` object contains the `mail` attribute, which will be the correlation key for loading Remedy accounts. The `mail` attribute also must be added to the schema map. Add the `email` attribute to the Waveset User Attribute side of the schema map, and `mail` to the Resource User Attribute side. `email` is a predefined Waveset attribute, so it is easier to user this attribute, rather than editing the User Extended Attributes or `UserUIConfig` configuration objects to include a `mail` attribute.

Waveset stores account IDs in the User object in the attribute `resourceAccountIds`. This is a multi-valued attribute, with each value taking the form `accountId@objectId`. You can create a rule that will compare the `EmployeeId` value from LDAP to the PeopleSoft `accountId` using the following rule:

Comparing `EmployeeId` value from LDAP to PeopleSoft accountId

```
<Rule subtype='SUBTYPE_ACCOUNT_CORRELATION_RULE' name='Correlate EmployeeId with accountId'>
   <cond>
      <ref>account.EmployeeId</ref>
      <list>
         <new class='com.waveset.object.AttributeCondition'>
            <s>resourceAccountIds</s>
            <s>startsWith</s>
            <concat>
               <ref>account.EmployeeId</ref>
               <s>@</s>
            </concat>
         </new>
      </list>
   </cond>
</Rule>
```

In this scenario, it is not necessary to add attributes to the User Extended Attributes or `UserUIConfig` configuration objects, because the `accountId` and email attributes are always available to the system.

▼ To Load LDAP Accounts

1. **From the Resources page in the Administrator Interface, select the LDAP resource from the New Resource pull-down menu. Then configure the adapter as follows:**

   a. **Add the EmployeeId and email Waveset User attributes.**

   b. **Make sure you do not delete the `accountId` Waveset user attribute from the schema map.**

   c. **Ensure that the identity template is correct.**

      See the online help and the Resource Reference for more information about configuring the adapter.

2. **Configure the reconciliation policy for the resource as follows.**

   a. **Set the Correlation Rule to Correlate EmployeeId with `accountId`.**

   b. **Set the following situation values:**

      Set the UNASSIGNED situation to “Link resource account to Identity Manager user”.

Set the UNMATCHED situation to an appropriate action. You might need to discuss with the PeopleSoft administrator about the possibility of adding users who are discovered on other resources. If you select the "Create new Waveset user based on resource account" option, the Waveset user will have, by default, an account name based on the LDAP `cn` attribute.

3 **Reconcile the LDAP resource.**

Results of This Scenario

In this scenario, accounts for William Cady, Josie Smith, and John P. Thomas will be in the UNMATCHED state. For Josie Smith, the employee ID values on the PeopleSoft and LDAP resources do not match. Because employee IDs are generated by PeopleSoft, then LDAP value is incorrect. Correct the mistake and reconcile again.

William Cady and John P. Thomas are not defined in PeopleSoft. As mentioned in step 2 of loading LDAP account procedure, you should consider whether the accounts need to be added to PeopleSoft.

Loading Remedy Users

The Remedy adapter does not have predefined account attributes. You must add these attributes to the schema map. Remedy uses integers to uniquely identify each attribute that it tracks. For example, the Remedy account ID might be assigned a value such as 1002000100. These Remedy attribute numbers must be added as Resource User Attributes on the schema map. At minimum, you must add the following Waveset User attributes:

- `accountId`
- `email` (the correlation key)

The USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR correlation rule will link the Remedy accounts to the Waveset accounts.

▼ To Load Remedy Accounts

1 **From the Resources page in the Waveset Administrator Interface, select the Remedy resource from the New Resource pull-down menu. Then configure the adapter as follows:**

At minimum, add the `accountId` and `email` Waveset User attributes. Other attributes can also be added.

See the online help and the Resource Reference for more information about configuring the adapter.

2 **Configure the reconciliation policy for the resource as follows.**

a. **Set the Correlation Rule to** `USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR`**.**

b. **Set the following situation values:**

   i. **Set the** `UNASSIGNED` **situation to "Link resource account to Identity Manager user".**

   ii. **Set the** `UNMATCHED` **situation to an appropriate action.**

c. **Reconcile the Remedy resource.**

Results of This Scenario

The Remedy accounts for William Cady, Eric D'Angelo, and John P. Thomas correlated successfully because the email addresses defined in LDAP and Remedy matched. The Remedy account for Robert Blinn did not correlate. The e-mail address on Remedy was an alias. The other users in this scenario do not have Remedy accounts.

Expedited Bulk Add Scenario

The following procedure illustrates how a few users can be quickly added to Waveset using Create actions. Any resources referenced in the CSV file must be defined within Waveset before the CSV file is loaded into the system.

▼ To Perform a Bulk Add

1 **Generate a CSV file that contains information needed to perform a Create bulk action. See** *Create Bulk Actions* **for more information about the format of the CSV file.**

2 **If your CSV file does not contain passwords, set the default Password Policy Options so that passwords are generated by the system. To do this, click the Configure tab, then Policies on the left. Click the Default Waveset Account Policy link. Then select the Generated option from Password Provided by drop-menu.**

3 **Create a new provisioning task based on the default Create User provisioning task.**

From the XML view, in the `TaskDefinition` tag, edit the value of the `resultLimit` parameter to 0. This value determines the number of seconds the results of the task is to be retained.

Then delete the following sections from the task:

```
<Activity id='1' name='Approve'>
...
</Activity>

<Activity id='3' name='Notify'>
...
</Activity>
```

Note – The activity calling to the provisioner must remain, or users will not be created properly.

Be sure to rename the task, to a value such as Fast Create User.

4. **Create a new user form based on the default User form.**

   From the XML view, replace the entire `<Form>... </Form>` structure with the following:

   ```
   <Form help='account/modify-help.xml'>
      <Field name="viewOptions.Process">
         <Expansion>
            <s>Fast Create User</s>
         </Expansion>
      </Field>
   </Form>
   ```

   Be sure to rename the user form, to a value such as Fast User Form.

5. **Create an Waveset account that will be used to load accounts into Waveset. Assign the user form created in "Expedited Bulk Add Scenario" on page 85 to this user.**

6. **Log in to Waveset using the account created in the previous step.**

7. **Run the Create bulk action.**

◆ ◆ ◆ CHAPTER 5

Data Exporter

This chapter describes the Data Exporter feature and provides information required to deploy it.

What is Data Exporter?

Waveset processes user account information on a wide range of systems and applications, providing a controlled, audited environment useful for making changes that remain in compliance with corporate policies. Waveset is a "data light" architecture. It locally stores a minimal amount of account information on the systems and applications that it manages and fetches the data from the actual system or application when necessary.

This architecture helps reduce data duplication and minimizes the risks of transferring stale data during provisioning operations, but there are times when having the account data stored locally is desirable. For example, being able to query account information without accessing the underlying system or application can bring significant performance improvements for some operations, such as identifying all accounts that have a specific attribute value. Typically, the use of system or application account data is related to reporting operations rather than provisioning operations, but in some cases the data does have value to the organization.

In addition to being a "data light" architecture, Waveset uses a "current data only" data model, which means it does not keep historical records (other than the audit and system logs). The advantage of this model is that the size of the operational repository tends to be proportional to the number of accounts, systems, and applications being managed. As a result, the provisioning system itself needs less maintenance. However, the data processed by Waveset may be valuable for historical processing.

For example, questions similar to the following rely on historical data:

- Who had access to system X between time A and B, and who approved of that access?
- How many provisioning requests have been processed in the last 48 hours, and how long did each request take?

Data Exporter allows you to selectively capture a large amount of the information processed by Waveset, including the account and workflow data necessary to answer questions like those listed above. Waveset produces this data in a form that can flow into a data warehouse to be further processed or used as a basis for queries and transformations using commercial database transformation, reporting, and analysis tools.

You are not required to export data from Waveset. If you do not need to track this type of the historical data, you are not required to keep it. If you require this data, you are free to establish your own data aging and retention policies without impact to Waveset.

Exportable Data Types

Data Exporter can export both persistent and transient data. Persistent refers to the data Waveset stores in the repository. Transient data is data that is either not stored in the Waveset repository by default, or data that has a lifecycle that precludes periodic fetching of changed records. Some types of data are both transient and persistent, such as Task Instances and WorkItems. These data types are considered transient because they are deleted by Waveset at times that are not externally predictable.

Waveset exports the following data types.

TABLE 5–1 Supported Data Types

Data Type	Persistence	Description
Account	Persistent	Record containing the linkage between a User and a ResourceAccount
AdminGroup	Persistent	A group of Identity Manager permissions available on all ObjectGroups
AdminRole	Persistent	The permissions assigned to one or more ObjectGroups
AuditPolicy	Persistent	A collection of rules evaluated against an Identity Manager object to determine complicance to a business policy
ComplianceViolation	Persistent	Tracks a User's non-compliance with an AuditPolicy
Entitlement	Persistent	A record containing the list of attestations for a specific User
LogRecord	Persistent	A record containing a single audit record
ObjectGroup	Persistent	A security container that is modeled as an organization
Resource	Persistent	A system/application on which accounts are provisioned
ResourceAccount	Transient	A set of attributes that comprise an account on a specific Resource
Role	Persistent	A logical container for access
Rule	Persistent	A block of logic that can be executed by Waveset
TaskInstance	Transient and persistent	A record indicating an executing or completed process

TABLE 5–1 Supported Data Types (Continued)

Data Type	Persistence	Description
User	Persistent	A logical user that includes zero or more accounts
WorkflowActivity	Transient	A single activity of an Waveset workflow
WorkItem	Transient and persistent	A manual action from an Waveset workflow

Data Exporter allows you to define strategies for exporting each type of data, depending on the exact needs of the warehouse. For example, some data types may need to export every change to an object while other data types may be satisfied with exporting at a fixed interval, potentially skipping intermediate changes to the data.

You can select which types will be exported. Once a type is selected, all new and modified instances of that type will be exported. Persistent data types can also be configured to export deleted objects.

Data Exporter Architecture

When Data Exporter is enabled, Waveset stores each detected change to a specified object (data type) as a record in a table in the repository. At a configurable interval for each data type, the system executes two queries that select the records to export.

- The first query looks for persistent objects of the specified type in the repository that have changed since the last export. The Warehouse Task exports these records, determines the timestamp of the most recently-changed record, and uses this value as the starting point the next time the query is run.
- The second query searches the queue table. It locates all records of the specified data type, exports them, then deletes the records from the queue. Any records added after the query completes will be exported at the next cycle.

The exported records are not ordered. However, there are fields in the exported data that allow a subsequent query of the warehouse to put the data in chronological order.

In a typical deployment, Data Exporter writes data to a set of staging tables. Waveset provides SQL scripts that define these tables for each type of supported database. You do not need to modify these tables, unless your Waveset deployment contains extended attributes that need to be exported. However, if you have extended attributes that will be exported, then you must customize your export schema and compile your own factory class for handling these attributes. For more information, see “Customizing Data Exporter” on page 97.

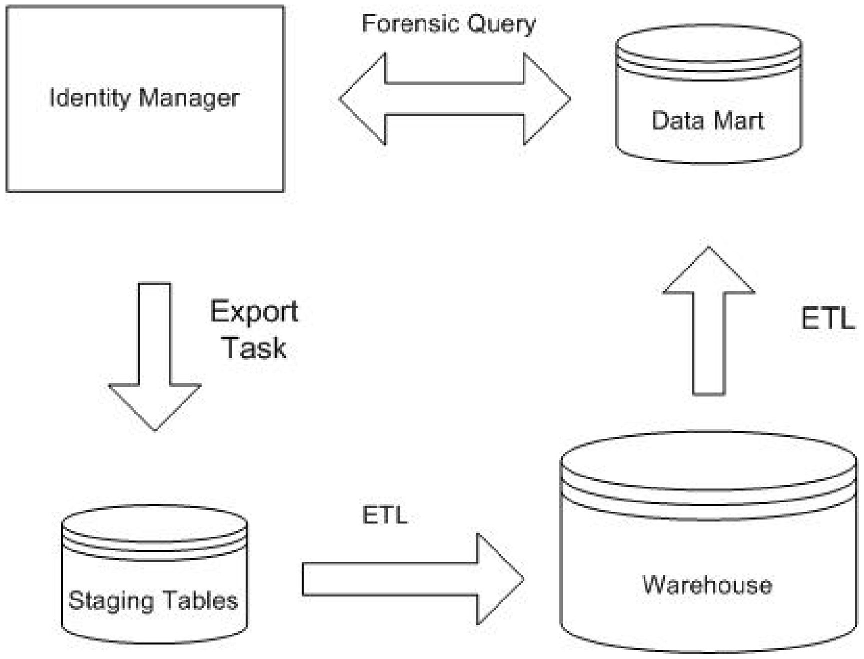
Exporting data to staging tables allows you to write your own Extract, Transform, and Load (ETL) infrastructure so that the data can be processed for storage in a data warehouse, and ultimately, in a datamart. Timestamp manipulation is a commonly-implemented transformation. The system uses the `java.sql.Timestamp` format of `YYYY-MM-DD hh:mm:ss`. Although the day of the week is not explicitly specified in the timestamp, it can be extracted using a transformation.

If you do not need to transfer information to a warehouse and datamart, then you can consider the staging tables to be the final destination. In this case, be sure to use the same connection information for read and write operations. See the Business Administrator's Guide for information about configuring Data Exporter.

Forensic queries allow Waveset to read data that has been stored in the data warehouse (or staging tables in a simple environment). They can identify users or roles based on current or historical values of the user, role, or related data types. A forensic query is similar to a Find User or Find Role report, but it differs in that the matching criteria can be evaluated against historical data, and because it allows you to search attributes that are of data types other than the user or role being queried. See the Business Administrator's Guide for information about defining forensic query.

The following diagram illustrates the data flow when Data Exporter is enabled.

FIGURE 5–1 Data Exporter Data Flow



Planning for Data Exporter

Before you begin deploying Data Exporter, you need to plan for the following:

- How much data will you export? The number of exported data types determines how many database tables will be required. If you choose to queue all changes to a data type, or even to export object deletions, the database requirements grow. See "Database Considerations" on page 91 for more information.
- Do you need to have a dedicated export server? Performance can be diminished if you export data on the same server that performs complex workflows. See "Export Server Considerations" on page 93 for more information.
- Do you have custom extended attributes that need to be exported? If yes, you must update the export schema and recompile the Warehouse Interface Code (WIC) and update the export schema. See "Customizing Data Exporter" on page 97 for more information.

Database Considerations

Data Exporter can export to any database that is supported as an Waveset repository. In addition, Data Exporter should also work with any RDBMS supported by Hibernate 3.2.

Hibernate Support

Data Exporter uses Hibernate 3.2 for the bi-directional mapping between Waveset Java objects and RDBMS tables. Waveset provides a set of files (one for each data type) that control the mapping between warehouse beans and RDBMS tables. These files are located in the `$WSHOME/exporter/hbm` directory.

See "Customizing Data Exporter" on page 97 for more details.

Hibernate uses C3P0 as its connection pool. C3P0 sends its log entries to the JRE logging system, which has INFO-level logging enabled by default. To restrict what is logged, add the following lines to the bottom of the `$JRE/lib/logging.properties` file:

```
com.mchange.v2.c3p0.impl.level=SEVERE
com.mchange.v2.c3p0.level=SEVERE
com.mchange.v2.log.level=SEVERE
```

Object/Relational Mapping

Waveset uses (Java) objects to perform its work, but when these objects are to be exported to a set of relational database tables, the objects must undergo a transformation commonly called *object/relational mapping*. This transformation is necessary because there are differences between the types of data that can be expressed in a RDBMS relationship and the types of data that can be expressed in an arbitrary Java object. For example, consider the following Java class:

```
class Widget {
  private String _id;
  private Map<String,Widget> _subWidgets;
  ...
}
```

This class presents a problem when expressed in relational terms, because the `_subWidgets` field is a nested structure. If you try decomposing two hierarchies of Widget objects that have shared subWidgets into a set of RDBMS tables, and delete one of the hierarchies, you quickly end up with a reference-counting problem.

To address the representational differences, Waveset places some constraints on what type of data can be exported. Specifically, the limit allows for the top-level Java object to contain scalar attributes, lists of scalar attributes, and maps of scalar attributes. In a few instances, Waveset needs a slightly richer expression, and to resolve these cases Waveset has introduced the `PseudoModel`. A PseudoModel is conceptually a data structure containing only scalar attributes. A top-level Java object can contain attributes that are PseudoModels or Lists of PseudoModels. PseudoModels are Waveset structures that cannot be extended. The following is an example of a PseudoModel.

```
class TopLevelModel
{
    private String _name;
    private List<PseudoModelPoint> _points;
}
class PseudoModelPoint
{
    private String _name;
    private String _color;
    private int _x;
    private int _y;
    private int _z;
}
```

Waveset can properly perform the object/relational transformation of `TopLevelModel` because `PseudoModelPoint` only contains scalar attributes. In query-notation, the color attribute of the PseudoModel is addressable as:

```
TopLevelModel.points[].color
```

When inspecting the Waveset Data Export schema, you will find a few PseudoModel types. These types represent some of the more complex data in the top-level export models. You cannot query for a PseudoModel directly because a PseudoModel is not exported directly. A PseudoModel is simply structured data held by an attribute of a top-level model.

Database Tables

The number of RDBMS tables defined in the warehouse DDL depends on the number of model types being exported, and what types of attributes each model is exporting. In general, each model requires three to five tables, with list/map valued attributes stored in their own table. The default DDL contains about 50 tables. After studying the export schema, you may choose to modify the Hibernate mapping files to exclude some attributes tables.

Space Requirements

The amount of space required in the exporter warehouse depends on

- Which objects are to be exported
- How long the records are to stay in the export warehouse
- How busy the Waveset servers are

WorkflowActivity and ResourceAccount are usually the highest-volume exported models. For example, a single workflow could contain multiple activities, and as each workflow is executed, Waveset could create dozens of new records to be written to the warehouse. Editing a User object may result in one ResourceAccount record per account linked to the User. TaskInstance, WorkItem and LogRecord are also high-volume models. A single Waveset server can produce over 50,000 object changes to be exported in one hour of operation.

Export Server Considerations

You should consider running the export task on a dedicated server, especially if you expect to export a large amount of data. The export task is efficient at transferring data from Waveset to the warehouse and will consume as much CPU as possible during the export operation. If you do not use a dedicated server, you should restrict the server from handling interactive traffic, because the response time will degrade dramatically during a large export.

The Export Task primarily performs input/output operations between the Waveset repository and the staging tables. The memory requirements of the export task are modest, although the memory needs increase as the number of queued records increases. The export task is typically constrained by the speed of the input/output and uses multiple concurrent threads to increase throughput.

Choosing the appropriate server requires experimentation. If the transfer rates of the input (Waveset repository) or the output (staging tables) are slow, the export task will not saturate a modern CPU. The query speed of the input path will not be an issue, as the export operation only issues a query at the beginning of the export cycle. The majority of the time is spent reading and writing records.

Waveset provides JMX MBeans to determine the input and output data rates. See Business Administrator's Guide for more information about these MBeans.

Loading the Default DDL

This section lists the commands needed to create a database and load the default Data Definition Language (DDL). The export DDL is generated by tools provided with Waveset to match the current export schema.

The `create_warehouse` scripts are located in the `$WSHOME/exporter` directory. Waveset also includes corresponding `drop_warehouse` scripts in the same directory.

DB2

Execute a script similar to the following as the system DBA. Be sure to create the `idm_warehouse` database and the `idm_warehouse/idm_warehouse user` before running the script.

```
CONNECT TO idm_warehouse USER idm_warehouse using 'idm_warehouse'
CREATE SCHEMA idm_warehouse AUTHORIZATION idm_warehouse
GRANT CONNECT ON DATABASE TO USER idm_warehouse
```

To load the DDL, add the following line to the `%WSHOME%\exporter\create_warehouse.db2` file:

```
CONNECT TO idm_warehouse USER idm_warehouse using 'idm_warehouse'
```

Then run the following command (assuming a Windows DB2 server):

```
db2cmd db2setcp.bat db2 -f create_warehouse.db2
```

MySQL

Execute a script similar to the following as the system DBA.

```
# Create the database (Schema in MySQL terms)
CREATE DATABASE IF NOT EXISTS idm_warehouse CHARACTER SET utf8 COLLATE utf8_bin;
# Give permissions to the "idm_warehouse" userid logging in from any host.
GRANT ALL PRIVILEGES on idm_warehouse.* TO idm_warehouse IDENTIFIED BY 'idm_warehouse';
# Give permissions to the "idm_warehouse" userid logging in from any host.
GRANT ALL PRIVILEGES on idm_warehouse.* TO idm_warehouse@'%' IDENTIFIED BY 'idm_warehouse';
# Give permissions to the "idm_warehouse" user when it logs in from the localhost.
GRANT ALL PRIVILEGES on idm_warehouse.* TO idm_warehouse@localhost IDENTIFIED BY 'idm_warehouse';
```

To load the DDL, execute the following command:

```
# mysql -uidm_warehouse -pidm_warehouse -Didm_warehouse < create_warehouse.mysql
```

Oracle

Execute a script similar to the following as the system DBA.

```
-- Create tablespace and a user for warehouse
CREATE TABLESPACE idm_warehouse_ts
   DATAFILE ’D:/Oracle/warehouse/idm_warehouse.dbf’ SIZE 10M
   AUTOEXTEND ON NEXT 10M
   DEFAULT STORAGE (INITIAL 10M NEXT 10M);
CREATE USER idm_warehouse IDENTIFIED BY idm_warehouse
   DEFAULT TABLESPACE idm_warehouse_ts
   QUOTA UNLIMITED ON idm_warehouse_ts;
GRANT CREATE SESSION to idm_warehouse;
```

To load the DDL, execute the following command

```
sqlplus idm_warehouse/idm_warehouse@idm_warehouse < create_warehouse.oracle
```

SQL Server

Execute a script similar to the following as the system DBA. Uncomment lines as necessary.

```
CREATE DATABASE idm_warehouse
GO
--For SQL Server authentication:
-- sp_addlogin user, password, defaultdb--For Windows authentication:
-- sp_grantlogin <domain\user>
--For SQL Server 2005:
--CREATE LOGIN idm_warehouse WITH PASSWORD = ’idm_warehouse’, DEFAULT_DATABASE = idm_warehouse sp_addlogin
’idm_warehouse’, ’idm_warehouse’, ’idm_warehouse’
USE idm_warehouse
GO
--For SQL Server 2005 SP2 create a schema - not needed in other versions:
--CREATE SCHEMA idm_warehouse
--GO
--For SQL Server 2005 SP2 use CREATE user instead of sp_grantdbaccess
--CREATE USER idm_warehouse FOR LOGIN idm_warehouse with DEFAULT_SCHEMA = idm_warehouse
sp_grantdbaccess ’idm_warehouse’
GO
```

To load the DDL, execute the following command:

```
osql -d idm_warehouse -U idm_warehouse -P idm_warehouse < create_warehouse.sqlserver
```

Upgrading Data Exporter

Data Exporter provides the means to periodically export data that is managed or has been processed by Waveset to a set of DBMS tables for further processing. The export process is intentionally open to customizations, some of which may require manual intervention for the proper behavior. The Waveset configuration objects that are relevant to Data Exporter are preserved and updated appropriately. However, some exporter customization is done to files within the web application, and these take special handling.

During the upgrade process, Waveset overwrites all unmodified Data Exporter files in the `$WSHOME` and `$WSHOME/exporter` directories. If you made changes to any Data Exporter files, then the upgrade process leaves your modified version in place and installs the newer version of the file in `$WSHOME/patches/Identity_Manager_8_1_0_0_`*Date*`/filesNotInstalled`. If you want to merge the new functionality with your customizations, you must do this manually.

Note that the following files in $WSHOME are often customized:

```
model-export.dtd
model-export.xml
model-export.xsl
exporter/exporter.jar
exporter/create_warehouse.*
exporter/drop_warehouse.*
exporter/hbm/*.hbm.xml
```

The upgrade steps you must perform vary depending on whether you customized Data Exporter in 8.0 and your plans for Data Exporter in 8.1

- If you customized Data Exporter for 8.0 and want to implement the 8.1 features:
  1. Drop the warehouse schema.
  2. Upgrade Waveset.
  3. Recreate the schema with the new DDL in the `$WSHOME/exporter` directory.

     There are no schema upgrade scripts that will allow the schema to be modified with data in place. Therefore, if you need to preserve the data, you must export and then import the data. The 8.1 warehouse schema is table and field compatible with the previous version, although 8.1 added new tables and new fields to existing tables. The field order was also changed. As a result, your export needs to be a data-only export, not a DDL and data export.
  4. Merge customizations with the new 8.1 exporter files. If `model-export.xml` was customized, rebuild the `exporter.jar` file.
  5. Load the new warehouse schema.
- If you customized Data Exporter for 8.0 and you do not want to implement the 8.1 features:

You can upgrade to 8.1 without performing any additional steps. However, if you upgrade to 8.1 Exporter but do not upgrade the warehouse DDL, the Warehouse Configuration page displays an error message that indicates the EXT_ADMINGROUP table is missing. This is an indication that the new 8.1 objects are in place, but the old 8.0 warehouse DDL is still loaded.

- If you did not customize Data Exporter for 8.0 and do not plan to implement the 8.1 features:

  1. Drop the warehouse schema.
  2. Upgrade Waveset.
  3. Load the new warehouse schema.

Note – Data in the warehouse is left untouched. You do not need to change the DDL if `model-export.xml` was customized. If `model-export.xml` was not customized, then you must load the new DDL.

After 8.1 is installed, if the 8.1 version of model-export.xml is in place, you can see the new data types and attributes by looking at the schema file at `http://`*server*`:`*port*`/idm/model-export.xml`. New types and attributes are flagged with the 8.1 release number.

Customizing Data Exporter

Data exporting has two levels of schema in effect, the internal (ObjectClass) and the external (Export) schemas. These schemas provide a data “interface” that can be proven to be compliant over multiple releases of the Waveset. Compliant means that the attribute names, data types, and data meanings will not change. An attribute may be removed, but the attribute name cannot be re-used to mean something different. Attributes may be added at any time. A compliant schema allows reports to be written against a version of the schema and run without modification against any later version.

The ObjectClass schema tells programs in the Waveset server what the data should look like, while the external schema tells the warehouse what the data should look like. The internal schema will vary from release to release, but the external schema will stay compliant across releases.

Waveset ObjectClass Schema

The ObjectClass schema can be extended for User and Role types, but otherwise cannot be changed. The ObjectClass schema is used by programs executing on the Waveset servers to provide access to the data objects themselves. This schema is compiled into Waveset and represents the data that is stored and operated on within Waveset.

This schema may change between versions of Waveset, but is abstract to the data warehouse because of the export schema. The ObjectClass schema provides a schema abstraction on top of the Waveset Persistent Object layer, which are the data objects stored in the Waveset repository.

Custom User and Role attributes, also known as extended attributes, are defined in the `IDMSchemaConfiguration` object. See Chapter 12, "Editing Configuration Objects," for information about adding extended attributes to the ObjectClass schema.

Export Schema

The export schema defines what data can be written to the warehouse. By default, it is limited to a subset of the ObjectClass schema, although the difference between the two is very small. The ObjectClass schema is represented by Java objects, but the export schema must have a bi-directional mapping between Java objects and RDBMS tables.

After you have added an extended attribute to the `IDMSchemaConfiguration` object, you must define the same attribute in the export schema, which is defined in the `$WSHOME/model-export.xml` file. Locate the Role or User model in this file and add a `field` element that defines the attribute. The `field` element can contain the following parameters.

TABLE 5–2 Export attribute parameters

Parameter	Description
name	The name of the attribute. This value must match the name assigned in the `IDMSchemaConfiguration` object.
type	The data type of the attribute. You must specify the full Java class name, such as `java.lang.String` or `java.util.List`.
introduced	Optional. Specifies the release that the attribute was added to the schema.
friendlyName	The label that is displayed on the Data Exporter configuration pages.
elementType	If the `type` parameter is `java.util.List`, then this parameter specifies the data type of the items in the list. Common values include `java.lang.String` and `com.sun.idm.object.ReferenceBean`.
referenceType	If the `elementType` parameter is `com.sun.idm.object.ReferenceBean`, then this parameter references to another Waveset object or pseudo-object.
forensic	Indicates the attribute is used to determine relationships. Possible values are `User` and `Role`.
exported	When set to false, the attribute is not exported. If you want to hide a default attribute from the Data Exporter data type configuration page, add `exported='false'` to the attribute definition. You must create a custom WIC library to be able to export an attribute in the default schema that has exporting disabled.

TABLE 5–2 Export attribute parameters *(Continued)*

Parameter	Description
queryable	When set to false, the field is not available for forensic queries.
max-length	The maximum length of a value.

The following example adds an extended attribute named `telno` to the export schema as part of the User model:

```
<field name='telno'
    type='java.lang.String'
    introduced='8.0'
   max-length='20'
   friendlyName='Telephone Number'>
   <description>The phone number assigned to the user.</description>
</field>
```

Modifying the Warehouse Interface Code

The Warehouse Interface Code (WIC) is provided in binary and source form in Waveset. Many deployments will be able to use the WIC code in binary form (no modifications), but some deployments may want to make other changes. The WIC code must implement two interfaces to be used for exporting, and a third interface to be used by the Forensic Query interface.

The default WIC implementation writes to a set of RDBMS tables. For many applications this is sufficient, but you could create custom WIC code to write the date to a JMS queue or to some other consumer.

The `com.sun.idm.exporter.Factory` and `com.sun.idm.exporter.Exporter` classes are used to export data. The export code is responsible for converting models (Java data objects) to a form suitable for storage. Typically, this means writing to a relational database. As a result, the WIC code is responsible for Object to Relational transformation.

The default WIC implementation uses Hibernate to provide the Object/Relational mapping. This mapping is controlled by the Hibernate `.hbm.xml` mapping files, which are in turn generated based on the export schema. Hibernate prefers to use a Java bean-style data object for its work, and has various get and set methods to accomplish this. The WIC code generates the corresponding Bean and hibernate files that match the export schema. If Hibernate provides the necessary mapping features, there may be no need to modify any WIC code manually.

The WIC files are located in the *InstallationDirectory*`/REF/exporter` directory.

Generating a New Factory Class

Waveset allows you to add custom User and Role attributes to the ObjectClass schema. These attributes, known as extended attributes, cannot be exported unless you also add them to the export schema, regenerate the Warehouse Interface Code (WIC), and deploy the code.

When extended attributes are added, you will need to edit the export schema control file and add the attributes. If attributes are to be excluded from the exporter, then you can simply mark the schema fields with `exported='false'` and regenerate the WIC code.

To modify the WIC code you will need the following installed on your system

- An installation of Waveset
- Java 1.5 SDK
- ant 1.7 (or later)
- A text editor

The steps required to export extended attributes are as follows:

▼ To Export Extended Attributes

1. **Get the WIC source code from the REF kit**

2. **Set the** `WSHOME` **environment variable to the installation directory of Waveset**

3. **Back-up the export schema control file** `$WSHOME/model-export.xml` **then edit it.**

4. **Change directories to the WIC source top-level directory. This directory should contain files named** `build.xml`**,** `BeanGenerator.java`**, and** `HbmGenerator.java`**.**

5. **Stop the application server.**

6. **Remove** `CLASSPATH` **from the environment.**

   Note – You must remove `CLASSPATH` from the environment before performing executing `ant rebuild` in the next step.

   ---

7. **Rebuild the WIC code with the** `ant rebuild` **command.**

8. **Deploy the modified WIC code to the application server with the** `ant deploy` **command.**

9. **Restart the application server.**

Note – If you change model-export.xml and rebuild the WIC as shown in the preceding steps, a new warehouse DDL is generated. You must drop the old tables and load the new DDL, which deletes any data that is already in the tables.

Adding Localization Support for the WIC

The export schema contains numerous strings that are displayed on the Data Exporter Type Configuration pages. Use the following procedure to display the strings in a language that is not officially supported:

▼ To Add Localization Support for the WIC

1. **Extract the contents of the `$WSHOME/WEB-INF/lib/wicmessages.jar` file.**

2. **Navigate to the `com/sun/idm/warehouse/msgcat` directory.**

3. **Translate the contents of `WICMessages.properties` file. Make sure the final results are saved in a file that contains the locale.**

   You do not need to save the message catalog to the System Configuration object.

Troubleshooting Data Exporter

The volume and variety of data flowing through the exporter increase the possibility of problems occurring during data export.

Beans and Other Tools

Data Exporter performance and throughput can be monitored through the JMX management beans provided in Waveset. To minimize the performance impact of exporting data, Waveset uses some memory-based queues that are volatile. If the server terminates unexpectedly, the data in these queues will be lost. You can monitor the size of these queues over a period of time to judge your exposure to this risk.

Model Serialization Limits

Data Exporter must queue some objects to ensure they are available for export at the appropriate time. Queuing these objects is done by Java serialization. However, it is possible to include data in an exported object that is not serializable. In this case, the exporter code should detect the non-serializable data and replace it with tokens that indicate the problem, allowing the rest of the object to be exported.

Repository Polling Configuration

Each type may specify an independent export cycle. The administrator interface provides an easy way to define the simpler cycles which will be sufficient for most purposes. However, the export cycles can also be specified in the native cron style, which supports even more flexibility.

Tracing and Logging

The default WIC code uses Hibernate to provide object/RDBMS mapping for the exported data objects, but using the Hibernate library means the tracing and logging is not fully integrated. The actual WIC code can be traced by using the com.sun.idm.warehouse.* package. However, enabling Hibernate logging requires a different technique.

To pass a Hibernate property to the code that initiates the Hibernate sessions, add an attribute to the `DatabaseConnection` configuration object. You must prefix the attribute name with an "X". For example, if the native property name is `hibernate.show_sql`, you must define it in the configuration object as `Xhibernate.show_sql`. The following example causes Hibernate to print any generated SQL to the application server's standard output.

```
<Attribute name='Xhibernate.show_sql' value='true'>
```

By default, Hibernate uses C3P0 for connection pooling. C3P0 uses the `java.logging` facility for its logging, which is controlled by the `$JRE/lib/logging.properties` file.

◆ ◆ ◆ CHAPTER 6

Integrating Waveset and SAP BusinessObjects Access Control

SAP BusinessObjects Access Control provides two interfaces for integration with Waveset. It provides a set of web services that Waveset can use to send and receive instructions needed to ensure a user has access to SAP applications. In addition, SAP Access Control can use SPML to send provisioning requests to Waveset so that SAP Access Control can enforce compliance and other features on resources that it does not directly support.

This document describes three options you can use to integrate SAP Access Control with Waveset.

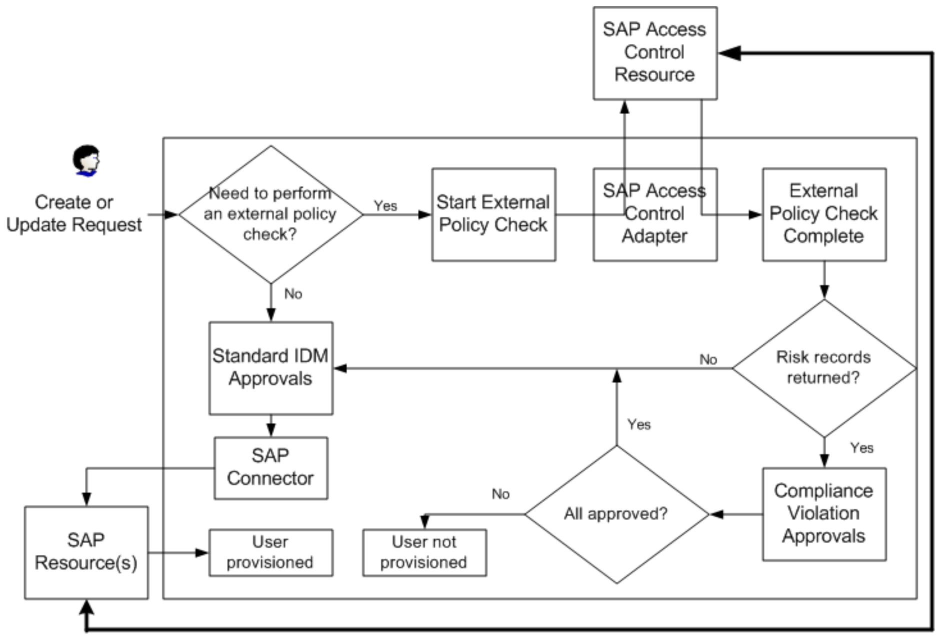
- **External Policy Check Integration**. Waveset submits requests using web services to SAP Access Control for Segregation of Duties (SoD) compliance. This option uses Waveset as the front-end interface and initiates requests into SAP Access Control for risk analysis, and if the risk analysis returns no violations or those violations are approved in Waveset, the request is provisioned using the SAP connector. This integration requires an SAP Access Control adapter to handle all web service requests and an SAP connector for provisioning.
- **External Resource Integration**. Waveset submits provisioning requests using web services to SAP Access Control. This option uses Waveset to initiate the workflow processes configured in SAP Access Control. SAP Access Control will then perform the provisioning operations after its workflow processes and approvals are complete. This integration requires an SAP Access Control adapter to handle all web service requests and an external resource for provisioning.
- **SPML 1.0 Integration**. SAP Access Control submits requests to Waveset for provisioning on non-SAP systems.

External Policy Check Integration

The external policy check integration allows an external policy product, such as SAP BusinessObjects Access Control, to perform a risk analysis when a user account is created or updated. Waveset gathers the relevant data and submits it to SAP Access Control. If risk analysis or SoD violations are returned, then the Waveset administrator determines whether to approve the violations or to reject the request entirely. If the administrator wants to change the SAP role assignments in a compliance violation, the request must be rejected and submitted again.

The following diagram illustrates the interaction between Waveset and SAP Access Control in this integration.

FIGURE 6–1 External Policy Check Integration


SAP Access Control Resource
Create or Update Request
Need to perform an external policy check?
Yes
Start External Policy Check
SAP Access Control Adapter
External Policy Check Complete
No
Standard IDM Approvals
No
Risk records returned?
Yes
Yes
SAP Connector
No
All approved?
Compliance Violation Approvals
SAP Resource(s)
User provisioned
User not provisioned

Waveset uses web services to communicate with the SAP Access Control resource. When Waveset receives a request to create or update a user, the Create User or Update User workflow launches the Check External Policy workflow.

The user view is queried for an `accounts[Lighthouse].properties.externalPolicy[`*ResourceName*`]` object. Each

ResourceName in the list specifies a resource on which to execute the external policy check. If objects for a resource do not exist, control is returned to the Create User or Update User workflow, respectively.

If objects exist, the Check External Policy workflow starts a deferred task that executes the external policy check on each resource. For SAP Access Control, this deferred task executes the risk analysis web service. While the risk analysis is executing, the workflow generates a temporary work item that suspends the create or update request until the external policy check is complete. The task runs synchronously, therefore the task monitors for the completion of the policy check and returns the risk analysis results to the workflow.

If the SAP Access Control resource returns one or more violations, the policy check creates a compliance violation work item. The Compliance Violation Owner rule determines who the owner of the work item and therefore who it is assigned to. By default, the rule assigns the work item to the manager defined in Waveset. If the rule cannot determine who to assign the rule to, it assigns it to the requestor.

Once the compliance violation has been assigned an owner, the owner can process the work item in the following ways:

- Approve the policy check. In this case, any violations are ignored, and Waveset continues to provision the user.
- Reject the request. The account is not provisioned on the SAP resource.
- If a previously-provisioned user has had a compliance violation check, any roles added in the future will generate compliance violation work items, regardless if the items would cause a compliance violation. If the request is rejected, only the requested roles will be rejected. The previous roles and provisioned users remain intact. This is because Waveset does not make any decisions or assumptions regarding the external policy service.

The `$WSHOME/sample/rules/SAPAccessControlRules.xml` file defines the SAPAccessControl rule library, which is used in the SAP Access Control and External SAP Access Control user forms and through a FormRef in the SAP Connector User Form. You may need to customize the following rules:

- Aggregate Roles
- Compliance Violation Owner
- Process Policy Result
- Risk Analysis Status

If you change the name of these rules, you must modify the prototype XML in the SAP Access Control adapter to match the renamed rule.

Note – The SAP Access Control Check Status rule is not used in an external policy check integration.

The forms used in this integration are defined in the `SAPAccessControlUserForm.xml` and the `SAPAccessControlCompViolForm.xml` files. The SAP Access Control user form aggregates the data required for performing a risk analysis. This data is placed in the `accounts[Lighthouse].properties.externalPolicies[ResourceName]` property in the User view. Also refer to the Javadocs for the classes in the `com.sun.idm.ws` package for detailed information about the SAP Access Control web services.

The SAP Access Control Compliance Violation Form controls the display of the compliance violation details on the Compliance Violations page. If you change the name of this form, you must modify the prototype XML in the SAP Access Control adapter to match the renamed form.

An SAP connector handles all provisioning requests. You must create an SAP connector for each SAP Access Control application name (SAP system) that SAP Access Control manages. SAP Access Control performs the risk analysis for all these systems, but the SAP connector performs the provisioning. Refer to the SAP connector documentation for detailed information.

Note – You cannot use an SAP adapter to provision. If you have an existing SAP adapter, you must migrate to the adapter to an SAP connector. Refer to Chapter 57, "Identity Connectors Overview," in *Oracle Waveset 8.1.1 Resources Reference* for information about migrating.

For information on this connector, and the Identity Connector project, refer to the https://identityconnectors.dev.java.net website.

The SAP Access Control AuditLog Report can be used to return information that is contained in the SAP Access Control audit log. This report is not installed by default, but is provided in the `$WSHOME/sample/sapreports.xml` file.

Supported Web Services

The external policy check integration supports the following web services for Risk Analysis and Remediation (RAR):

- SAPGRC_AC_IDM_SUBMITREQUEST
- SAPGRC_AC_IDM_RISKANALYSIS

In addition, the SAP Access Control AuditLog Report uses the SAPGRC_AC_IDM_AUDITTRAIL web service.

Implementing a External Policy Check Integration

Use the following guidelines to implement the external policy check integration in an environment in which you have not integrated Access Enforcer 5.2 or earlier. Before beginning this procedure, you must set up SAP Access Control as described in the “Access Control and Identity Manager” Integration chapter in the SAP GRC Access Control Configuring SAP with Release 5.3 guide, which is available from SAP. If you have previously integrated Access Enforcer, see “Migrating from Access Enforcer Version 5.1 or 5.2” in this document.

Note – Disable the autoprovisioning feature within SAP Access Control for the external policy check integration. Otherwise, the user will be provisioned by Access Control, and not by Waveset. If you want autoprovisioning, then implement the external resource integration.

---

1. Create an SAP Access Control adapter.
2. Create an SAP connector. Be sure to configure the following parameters.
   a. Select the Configure Policy Resource check box.
   b. Select the SAP Access Control resource from the Policy Resource menu.
   c. Select the SAP Access Control application associated with this connector from the Instance ID menu.

   Repeat this step for each SAP Access Control application.
3. Copy the following files and make any required modifications. Be sure to save the original files.
   - `$WSHOME/sample/rules/SAPAccessControlRules.xml`
   - `$WSHOME/sample/forms/SAPAccessControlUserForm.xml`
   - `$WSHOME/sample/forms/SAPAccessControlCompViolForm.xml`

   Then import the modified files.
4. The SAP connector glue code contains a sample SAP Connector User Form (`SAPUserForm.xml`). This file contains commented code that needs to be enabled. Delete the comment marks around the Include statement near the top of the file. Then delete the comment marks around the Field that contains the FormRef to the SAP Access Control User Form. Then import this file.
5. Configure the User Deferred Task Scanner to run periodically. The default interval is 1 hour. To obtain the risk analysis results as quickly as possible, the interval for the deferred task should be set as low as possible.

Managing Compliance Violations

View compliance violations from the Work Items area of the administrator interface. Selecting the Compliance Violations tab in the Work Items area lists all the violations requiring approval or rejection for the current administrator. From this page, you can also list compliance violations for all of your direct reports and for specified users for which you have direct or indirect control.

The following are possible responses to a compliance violation:

- Approve. Accepts the risk displayed in the compliance violation. This will probably generate a risk violation on the external system.
- Reject. Does not accept the risk. The provisioning operation ends.
- Refresh. Updates the list of compliance violations.
- Forward. Enables you to specify another recipient to review the compliance violation.

Click the Request link to view detailed information about the compliance violation. The risk code, SAP application, description, severity level, violation count, and violation details are displayed. Refer to the SAP Access Control documentation for more information about these fields.

External Resource Integration

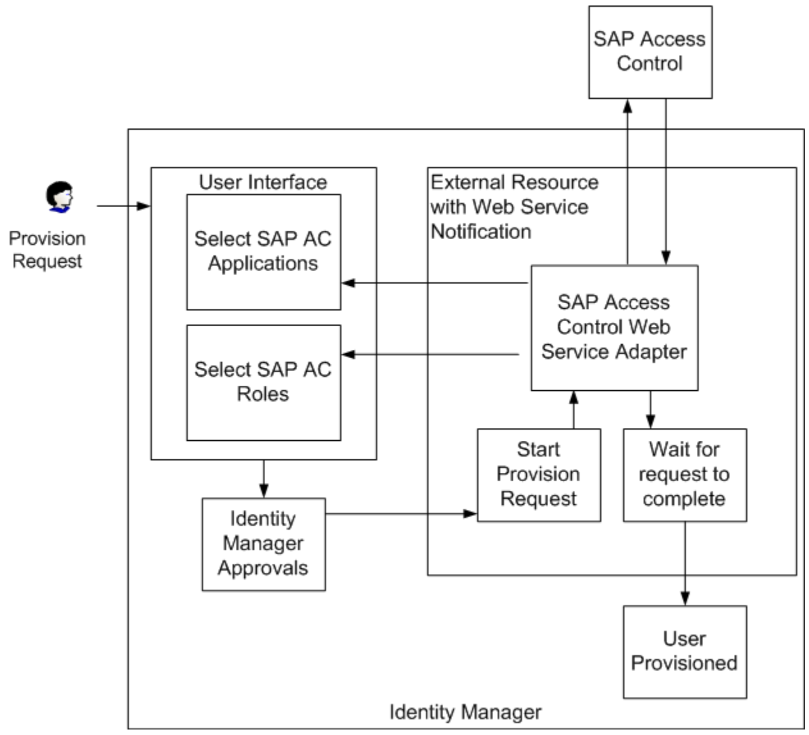
The external resource integration is provided to accommodate customers who already have an investment in an SAP Access Enforcer 5.1 or 5.2 custom workflow. The external resource provides the best way to integrate with this type of asynchronous functionality because in this configuration, SAP Access Control is considered a black box and as disconnected until the pending provisioning operation has been performed.

SAP Access Control can be considered a write-only resource because it has no user store of any kind. The external resource data store contains the user data sent to SAP Access Control on each request. This data store provides a way for Waveset to read the data that was last sent to Access Control. However, the data is not necessarily synchronized with SAP Access Control back-end systems. It only provisions the users to its configured SAP systems (applications).

Note – This integration does not perform an external policy check or generate compliance violations.

The following diagram illustrates the interaction between Waveset and SAP Access Control in this integration.

FIGURE 6–2 External Resource Integration


SAP Access Control
Provision Request
User Interface
Select SAP AC Applications
Select SAP AC Roles
External Resource with Web Service Notification
SAP Access Control Web Service Adapter
Start Provision Request
Wait for request to complete
Identity Manager Approvals
User Provisioned
Identity Manager

The external resource must be configured with a database user store (not LDAP) that contains the attribute values being provisioned. In addition, it must be configured to perform notifications by web services. To do this, set the Provisioner Notification Type parameter to Web Service. When the external resource is configured in this way, you must specify a rule to determine the status of web service requests. The `$WSHOME/sample/rules/SAPAccessControlRules.xml` file provides the SAP Access Control Check Status rule to fulfill this requirement.

The User Deferred Task Scanner polls SAP Access Control for the status of the provisioning request sent to SAP Access Control. When the request is complete, Waveset updates the workitem to indicate that the provisioning request is no longer pending, setting the status of the associated provisioning request work item to either completed or not complete.

The SAP Access Control AuditLog Report can be used to return information that is contained in the SAP Access Control audit log. This report is not installed by default, but is provided in the `$WSHOME/sample/sapreports.xml` file.

Supported Web Services

The external resource integration supports the following web services for Compliant User Provisioning (CUP):

- SAPGRC_AC_IDM_REQUESTSTATUS
- SAPGRC_AC_IDM_ROLEDETAILS
- SAPGRC_AC_IDM_SEARCHROLES
- SAPGRC_AC_IDM_SELECTAPPLICATION
- SAPGRC_AC_IDM_SUBMITREQUEST

In addition, the SAP Access Control AuditLog Report uses the SAPGRC_AC_IDM_AUDITTRAIL web service.

Implementing a New External Resource Integration

Use the following procedure to integrate Waveset with SAP Access Control using an external resource.

1. Create an SAP Access Control Web Service adapter.
2. Import the following files:
   - `$WSHOME/sample/rules/SAPAccessControlRules.xml`
   - `$WSHOME/sample/forms/ExternalSAPAccessControlUserForm.xml`
3. Create an external resource. Refer to "Understanding and Managing External Resources" in *Oracle Waveset 8.1.1 Business Administrator's Guide* for more information. The resource must have a database data source with Web Service notification. Be sure to follow these steps:
   a. Select Web Service as the Provisioner Notification Type.
   b. Select the desired SAP Access Control resource.
   c. Select either the SAP Access Control Check Status Rule or your customized rule to perform the status check. The SAP Access Control Check Status Rule is provided in the `SAPAccessControlRules.xml` file.
   d. Select the Provisioning Request form for SAP Access Control. The recommended form is the SAP Access Control Provisioning Request Form, but you may also select a customized form.
4. The account attributes page will be populated with attributes from the SAP Access Control resource, but they cannot be edited. If you make any changes to the attributes on the SAP Access Control resource, load the external resource account attributes page and click the Save button. Refer to the SAP Access Control adapter documentation for a list of default attributes.
5. Merge the External SAP Access Control User Form with your version of the Tabbed User Form.

6. Configure the User Deferred Task Scanner to run periodically. The interval for this task will determine how often the status of the provisioning request will be checked. The default interval is 1 hour.

Migrating from Access Enforcer Version 5.1 or 5.2

Upgrading from an SAP Access Enforcer resource to an external resource that connects to SAP Access Control 5.3 requires multiple tasks. Read the following procedure carefully.

Waveset provides a sample workflow, *InstallDir*`/sample/wfexternalmigration.xml`, for the purpose of migrating users from SAP Access Enforcer to SAP Access Control 5.3. It must be customized according to your environment and configuration.

The workflow requires the following account attributes:

- acRequestorId
- acRequestorFirstname
- acRequestorLastname
- acRequestorEmail
- acManagerId
- acManagerFirstname
- acManagerLastname
- acManagerEmail
- acPriority
- acUserId
- acApplications
- acRole

You may add attributes as needed.

This workflow currently displays the result of each user update. The current way around this is to modify the `taskLaunch.jsp` so that instead of displaying the results, the page is redirected back to the `taskList.jsp` page.

1. Determine the attributes that need to be migrated and update the workflow defined in the *InstallDir*`/sample/wfexternalmigration.xml` file.
2. Import the updated workflow.
3. Create an external resource for the SAP Access Control 5.3 integration. Ensure that the External Resource user store is configured properly. Refer to "Understanding and Managing External Resources" in *Oracle Waveset 8.1.1 Business Administrator's Guide* for more information.

Note – The SAP Access Control adapter, which uses Metro for Web Service communication, can not be instantiated in some application servers together with the Access Enforcer resource adapter, which uses Apache Axis for Web Service Communication.

---

4. Run the Sample SAP Access Enforcer To Access Control Migration server task. The system should not be in use by anyone else during this process..
5. Remove Apache Axis from application server.
6. Download and install Glassfish Metro 1.5 from the following location: https://metro.dev.java.net/1.5/
7. Remove the Access Enforcer resource from Waveset.
8. Create the SAP Access Control resource.
9. Modify the external resource to use SAP Access Control web services.

SPML 1.0 Integration

The integration scenario in which SAP BusinessObjects Access Control is used as the leading provisioning system and Waveset is used for provisioning to non-ERP systems requires specific configuration on both SAP Access Control and Waveset. The SAP Access Control configuration steps are part of the *SAP GRC Access Control 5.3 Configuration Guide*, chapter "Sending Provisioning Request to an IdM System".

SPML Configuration

Waveset provides an SPML schema that defines the object class and extended request information needed to configure SAP Access Control integration. The schema is defined in *InstallDir*`/sample/spml.xml`. You must edit this file before loading it into Waveset. Specifically, you must uncomment the IDMperson object. This object includes a change to the standard person object, as described in the comments contained in the file.

After this task has been performed, the file must be imported and the configuration finished following the steps in *Oracle Waveset 8.1.1 Web Services*.

Note – SAP Access Control tries to use HTTP basic authentication when it creates a connection to the Waveset system. If the container in which the Waveset server runs does not support basic authentication, a connection will be established without authentication. The user name and password specified in SAP Access Control as part of the connector cannot be used to create an Waveset session. No request will be processed by Waveset without a valid session. This means that you must configure the proxy user and password for the SPML interface as described in "Editing the Waveset.properties File" in *Oracle Waveset 8.1.1 Web Services*. The proxy user will allow anyone access to the SPML interface and care should be taken to shield the URL from misuse.

SAP Access Control Configuration Information

The following table lists actions and their corresponding parameter names and values. These must be set in the SAP Access Control connector configuration.

Action	Parameter Name	Parameter Value
Create User	SCHEMA_ID	standard
	CREATE_USER:OC	IDMperson
	CREATE_USER:options.AllowPasswordGeneration	true
	CREATE_USER:options.onlyResourcesUserPassword Required	true
Change User	CHANGE_USER:OC	IDMperson
Delete User	DELETE_USER:OC	IDMperson
Assign Roles	ASSIGN_ROLES:OC	IDMperson
	ROLE	roles
Lock User	LOCK_USER:EXT	disableUser
Unlock User	UNLOCK_USER:EXT	enableUser
Audit Logs	not configurable	not applicable
	AUDIT_TYPE	statusrequest
Reset Password	RESET_PASSWORD:EXT	resetUserPassword
Search Password	SEARCH_PASSWORD:EXT	launchProcess
	SEARCH_PASSWORD:process	SPML Decrypt Password
	SEARCH_PASSWORD:taskName	Decrypt Password

Action	Parameter Name	Parameter Value
Search	SEARCH_CRITERIA	identifier

SAP Access Control currently does not support filtering the SPML attributes defined in the schema based on the object class. When you create the mapping for the SAP Access Control connector, all attributes are displayed, even the attributes that are not part of the object class used. During the fields mapping SAP Access Control sends a SchemaRequest to Waveset to allow you to map the attributes for the connector in SAP Access Control. By default, the Waveset schema contains multiple object classes, and you will see attributes that are not valid for the object class you have configured. There are two possible workarounds for this:

- Reduce the SPML schema on the Waveset server temporarily to use just the object class and attributes needed. When the SAP Access Control server is configured you should reload the original schema again. Changes to the SPML schema require a restart of the server.
- Use a printout of the schema and mark the available attributes for the object class used and do not rely on the attributes presented in the drop down of the SAP Access Control user interface.

The following table lists field mappings for the SAP Access Control connector. This is not a complete list of all the fields which could be mapped.

Access Control Field	Application Field
Email Address - STANDARD	email
User FName - STANDARD	gn
User ID - STANDARD	accountId
User LName - STANDARD	sn

These application fields are the SPML schema attribute names. These names do not have to correspond with internal Waveset attribute names. In the SPML configuration, these names can be mapped using a form to internal Waveset attribute names.

Note – The SAP Access Control connector must not be configured to run over HTTPS.

SPML Request Behavior

An action on a user is a request SAP Access Control sends to Waveset using SPML. The SPML request is a batch request type. One request in SAP Access Control can consist of multiple steps, each an individual SPML request inside the batch, that Waveset must process. This behavior is not configurable on the SAP Access Control side, but it has implications for processing the request on both systems.

Waveset processes the individual requests inside the batch in the order that they are received. A status is returned for each of the requests processed: success, failure, or pending. A failure will show up as an error in SAP Access Control, marking the action appropriately. Any request that requires an approval or other manual action in Waveset creates a background process. The status that is returned for these requests will be pending. Any pending requests will be interpreted as a failed request by SAP Access Control. Processing of the request on the Waveset side is not impacted by this.

SAP Access Control does not maintain state of the individual requests sent inside the batch and marks the whole request as a failure if at least one request returns a status of failure. The approval or manual action on Waveset should complete as normal. If the changes were approved in Waveset, the request must then be resubmitted or reapproved in SAP Access Control. This action will generated a new batch request which will retry each individual request. In this case, the audit log entries in SAP Access Control could show inconsistent information about the user state on Waveset. The final state in SAP Access Control should be correct after the batch request is resubmitted.

User Create Password Notification

When a user is created in Waveset using SPML from SAP Access Control, Waveset generates the password. SAP Access Control does not provide a password in the initial create request, but it tries to retrieve the password from Waveset in clear text after the user has been created. To satisfy SAP Access Control process requirements, the Create User workflow generates the password and informs the user by sending an email notification. The workflow sends a notification only if the user has no password and if the `AllowPasswordGeneration` parameter is set to true.

Waveset uses the Generated Password Notice email template to send out this notification.

Audit Log Support

SAP Access Control can request audit log entries from the Waveset system, but these entries are not automatically enabled. The default requests rely on audit entries that have the SPML request ID as part of the attributes. Normal audit log entries neither contain nor have access to the SPML request ID.

To enable this feature, add the following attribute to the `Waveset.properties` file:

```
soap.auditlog=true
```

With this setting, Waveset adds request IDs to all subsequent audit events. It does not change any of the currently logged entries. The number of logged audit events will not increase or decrease as a result of this change. However, the same audit entry can be logged twice: once with the request ID, and once without.

SPML requests that contain one request ID and multiple actions are called batch requests. SAP Access Control often uses them during updates of user objects. These batch requests cause multiple audit log entries to be generated with the same SPML request ID. The Waveset Administrator interface displays these log entries as separate entries. If the extended SPML request auditlog is used to retrieve the audit log entries, only one entry will be returned. This behavior might change in future releases.

CHAPTER 7

Configuring User Actions

This chapter details how to add custom tasks to the Waveset Administrator Interface and configure user actions that you can execute from two areas of the interface:

- User Account Search Results page
- User applet on the Accounts page

Note – To add a custom task, you must edit an existing `TaskDefinition` You can use the Identity Manager IDE to view and edit task definitions. Instructions for installing and configuring the Identity Manager IDE are provided on `https://identitymanageride.dev.java.net`.

Adding Custom Tasks

Follow these general steps to add custom tasks:

- Set up authorization for the task
- Add the task to the repository

Setting Up Custom Task Authorization

Typically, you set authorization for custom tasks to restrict access to the task to a certain set of administrators.

▼ To Set Up Authorization

1. **Add a new authorization type (`AuthType`) to the repository for the task**

2. **Create a new `AdminGroup` (capability) for the task**

3. **Grant the new capability to one or more administrators**

Step 1: Create an AuthType

The new authorization type you create should extend the existing `TaskDefinition`, `TaskInstance`, and `TaskTemplate` AuthTypes. To add the authorization type, edit the Authorization Types Configuration object in the repository and add a new authorization type element for your task.

Use the `<AuthType>` element to create a new authorization type. This element has one required property: `name`. The example below displays the correct syntax for an `<AuthType>` element.

After creating the authorization type, you must edit the Authorization Types Configuration object in the repository, and add the new `<AuthType>` element.

The following example shows how to add a custom task to move multiple users into a new organization.

EXAMPLE 7–1 Moving Multiple Users into a New Organization

```
<Configuration name='AuthorizationTypes'>
   <Extension>
      <AuthTypes>
         <AuthType name='Move User' extends='TaskDefinition,TaskInstance,TaskTemplate'/>
      </AuthTypes>
   </Extension>
</Configuration>
```

Step 2: Create an AdminGroup

Next, create an `AdminGroup` that grants `Right.VIEW` for the newly created AuthType. To do this, you must create an XML file with the new administrator group, and then import it into the Waveset repository.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Waveset PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Waveset>
   <AdminGroup name='Move User' protected='true' description='UI_ADMINGROUP_MOVE_USER_DESCRIPTION'
displayName='UI_ADMINGROUP_MOVE_USER' >
      <Permissions>
         <Permission type='Move User' rights='View'/>
      </Permissions>
      <MemberObjectGroups>
         <ObjectRef type='ObjectGroup' id='#ID#All' name='All'/>
      </MemberObjectGroups>
   </AdminGroup>
</Waveset>
```

The `displayName` and `description` attributes are message catalog keys. If these are not found in a message catalog, they are displayed as they are found in the attributes. If message catalog keys are used, you must add the messages either into `WPMessages.properties` or a custom message catalog.

Step 3: Grant Capabilities to Administrators

Finally, you must grant administrators access to execute the newly defined task. You can accomplish this in one of two ways:

- Directly assign the new capability.
- Add the new capability to an Admin Role (either directly or by using a capabilities rule), and then assign it.

Adding a Task to the Repository

After you set up task authorization, you can add the task to the repository. The task is a typical `TaskDefinition` that can be defined through the Identity Manager IDE or imported as XML. For example, a task to change the organization for multiple users would resemble the following example (which is included in the `samples` directory).

EXAMPLE 7–2 Changing the Organization for Multiple Users

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE TaskDefinition PUBLIC 'waveset.dtd' 'waveset.dtd'>
<!-- MemberObjectGroups="#ID#Top" authType="Move User" name="Change Organizations"
taskType="Workflow" visibility="runschedule"-->
<TaskDefinition authType='MoveUser' name='Change Organizations' taskType='Workflow'
executor='com.waveset.workflow.WorkflowExecutor' suspendable='true' syncControlAllowed='true' execMode='sync'
execLimit='0' resultLimit='0' resultOption='delete' visibility='runschedule' progressInterval='0'>
   <Form name='Change Organization Form' title='Change Organization Form'>
      <Display class='EditForm'/>
      <Include>
         <ObjectRef type='UserForm' name='User Library'/>
         <ObjectRef type='UserForm' name='Organization Library'/>
      </Include>
      <FieldRef name='namesList'/>
      <FieldRef name='orgsList'/>
      <FieldRef name='waveset.organization'/>
   </Form>
   <Extension>
      <WFProcess name='Change Organizations' title='Change Organizations'>
         <Variable name='waveset.organization'/>
         <Variable name='userObjectIds' input='true'>
            <Comments>The names of the accounts to change the organization on.</Comments>
         </Variable>
         <Activity id='0' name='start'>
            <ReportTitle>
               <s>start</s>
            </ReportTitle>
            <Transition to='Process Org Moves'/>
         </Activity>
         <Activity id='1' name='Process Org Moves'>
            <Action id='0' process='Move User'>
               <Iterate for='currentAccount' in='userObjectIds'/>
               <Argument name='userId' value='$(currentAccount)'/>
               <Argument name='organizationId' value='$(waveset.organization)'/>
             </Action>
```

EXAMPLE 7–2 Changing the Organization for Multiple Users *(Continued)*

```
            <Transition to='end'/>
        </Activity>
        <Activity id='2' name='end'/>
     </WFProcess>
  </Extension>
  <MemberObjectGroups>
     <ObjectRef type='ObjectGroup' id='#ID#Top' name='Top'/>
  </MemberObjectGroups>
</TaskDefinition>
```

Note these features of the preceding example:

- The task's `authType` attribute is set to Move User. This will restrict access to this task to users that are assigned the capability to execute this authorization type.
- The form contains FieldRefs to `namesList` and `orgsList`. These fields are defined in the User Library and Organization Library, respectively. Including these fields will display lists of the names of all selected users and all selected organizations. *For potentially dangerous tasks, you should include one or both of these fields so the user is aware of the potential effects of running the task.*
- The task has an input variable named `userObjectIds`. This variable contains a list of the names or IDs of the users selected in the User Account Search Results page or in the user applet on the Accounts page. Iterate over this variable to perform the desired action on all selected users.

The following table lists the variables that are available for input to the task.

TABLE 7–1 Task Variables

Variable	Description
`userObjectIds`	List of IDs of the selected users. Available from the User Account Search Results and Accounts pages. When invoked from the User Account Search Results page, this list contains the names of the selected users.
`userNames`	List of names of the selected users. Available from the User Account Search Results and Accounts pages.
`orgObjectIds`	A List of IDs of the selected organizations. Available only from the Accounts page.
`orgNames`	A List of names of the selected organizations. Available only from the Accounts page.

To enable this workflow, you must also add to the repository a sub-process to change a user's organization, as shown in the following example.

EXAMPLE 7–3 Changing a User's Organization

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<!-- MemberObjectGroups="#ID#Top" configType="WFProcess" name="Move User"-->
<Configuration name='Move User' createDate='1083353996807'>
   <Extension>
      <WFProcess name='Move User' title='Move User'>
         <Variable name='userId' input='true'>
            <Comments>The accountId of the user to move.</Comments>
         </Variable>
         <Variable name='organizationId' input='true'>
            <Comments>The ID of the organization to move the user into.</Comments>
         </Variable>
         <Activity id='0' name='Start'>
            <Transition to='Update Organization'/>
         </Activity>
         <Activity id='1' name='Update Organization'>
            <Action id='0' process='Update User View'>
               <Argument name='accountId' value='$(userId)'/>
               <Argument name='updates'>
                  <map>
                     <s>waveset.organization</s>
                     <ref>organizationId</ref>
                  </map>
               </Argument>
            </Action>
            <Transition to='End'/>
         </Activity>
         <Activity id='2' name='End'/>
      </WFProcess>
   </Extension>
   <MemberObjectGroups>
      <ObjectRef type='ObjectGroup' id='#ID#Top' name='Top'/>
   </MemberObjectGroups>
</Configuration>
```

Configuring User Actions

You must configure definitions for the buttons and actions menu selections that initiate custom actions. Definitions for the buttons and actions menu items that appear on the User Account Search Results and Accounts pages are contained in the User Actions Configuration configuration object.

Do not directly edit the User Actions Configuration object. Rather, best practice for configuring user actions is to:

- Copy the User Actions Configuration configuration object into a new configuration object.
- Modify the System Configuration object to point to the new configuration object.

▼ To Configure User Actions

1 Copy the User Actions Configuration configuration object into a new XML file.

2 Change the name of the new object to My User Actions Configuration.

3 Make any desired modifications to My User Actions Configuration.

4 Import the XML file into Waveset from the Import Exchange File page

5 Modify SystemConfiguration to change the `userActionsConfigMapping` **attribute's value to My User Actions Configuration**

The configuration object consists of these configuration sections.

Attribute	Description
`findUsersButtons`	Contains a list of button definitions for the Administrator Interface User Account Search Results page.
`userApplet.userMenu`	Contains a list of menu item definitions for the user actions menu. This menu displays when you right-click a user in the applet on the Administrator Interface Accounts page.
`userApplet.organizationMenu`	Contains a list of menu item definitions for the organization actions menu. This menu displays when you right-click an organization in the applet on the Accounts page.

Each section contains a list of user actions to display in the interface. The button and menu configuration items have the same basic properties. Both include several extensions unique to the interface.

The following excerpt is an example of the user action configuration customized to add the Change Organization task to each list.

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Waveset PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Waveset>

<Configuration name='My User Actions Configuration'>
  <Extension>
    <Object>
      <!-- Buttons for the find users results page. -->
      <Attribute name='findUsersButtons'>
        <List>
          <Object>
            <Attribute name='textKey' value='UI_NEW_LABEL' />
            <Attribute name='commandName' value='New' />
            <Attribute name='requiredPermission'>
              <Object>
```

```
                <Attribute name='objectType' value='User' />
                <Attribute name='rights' value='Create' />
              </Object>
            </Attribute>
            <Attribute name='alwaysDisplay' value='true' />
          </Object>
          ...
          <Object>
            <Attribute name='textKey' value='UI_CHANGE_ORGANIZATIONS_LABEL' />
            <Attribute name='commandName'
value='Change Organizations' />
          </Object>
        </List>
</Attribute>
      <Attribute name='userApplet'>
        <Object>
          <!-- The menu to display when a user is selected. -->
          <Attribute name='userMenu'>
            <List>
              <Object>
                <Attribute name='textKey'
value='UI_ACCT_JAVA_MENU_NEW_ORG' />
                <Attribute name='commandName'
value='New Organization' />
                <Attribute name='requiredPermission'>
                  <Object>
                    <Attribute name='objectType' value='ObjectGroup' />
                    <Attribute name='rights'     value='Create' />
                  </Object>
                </Attribute>
              </Object>
              ...
              <Object>
                <Attribute name='separator' value='separator' />
              </Object>
              <Object>
                <Attribute name='textKey'
value='UI_CHANGE_ORGANIZATIONS_MENU_LABEL' />
                <Attribute name='commandName'
value='Change Organizations' />
              </Object>
            </List>
</Attribute>
          <!-- The menu to display when an organization is selected. -->
          <Attribute name='organizationMenu'>
            <List>
              <Object>
                <Attribute name='textKey'
value='UI_ACCT_JAVA_MENU_NEW_JUNCTION' />
                <Attribute name='commandName'
value='New Directory Junction' />
                <Attribute name='requiredPermission'>
                  <Object>
                    <Attribute name='objectType' value='ObjectGroup' />
                    <Attribute name='rights'     value='Create' />
                  </Object>
                </Attribute>
                <Attribute name='orgTypes' value='normal,dynamic' />
              </Object>
              ...
              <Object>
                <Attribute name='separator' value='separator' />
              </Object>
              <Object>
                <Attribute name='textKey'
value='UI_CHANGE_ORGANIZATIONS_MENU_LABEL' />
                <Attribute name='commandName'
value='Change Organizations' />
              </Object>
            </List>
          </Attribute>
        </Object>
      </Attribute>
    </Object>
  </Extension>
  <MemberObjectGroups>
    <ObjectRef type='ObjectGroup' name='All'/>
</MemberObjectGroups>
</Configuration>
</Waveset>
```

User action definitions support these core attributes.

Attribute	Description
`textKey`	Message catalog key for the text of the button or menu item.
`commandName`	Name of the command to execute. This can be a command that is natively supported (such as New or Delete User), or the name of a `TaskDefinition` to execute.
`requiredPermission.objectType`	Type of object that the rights are required on in order to display this item. This is applicable only for natively supported commands. Task Definitions should use AuthTypes for controlling access.
`requiredPermission.rights`	Comma-separated list of `Right` names required on the specified objectType to display this item. This is applicable only for natively supported commands. Task Definitions should use AuthTypes for controlling access.
`alwaysDisplay`	Optional. Specifies whether to always display this button. If set to a value of `true`, the button is displayed even if user search returns no results. The default value for this attribute is `false`. Applies to `findUsersButtons` section only.

User actions definitions in the `userApplet` section also support the attributes in the following table.

Attribute	Description
`orgTypes`	Comma-separated list of organization types for which to display the item in the organization menu. Possible values are normal, dynamic, and virtual for normal organizations, dynamic organizations, and virtual organizations, respectively. If this attribute is not specified, the menu item is displayed for all organization types.
`separator`	Special item in the format `<Object><Attribute name=’separator’ value=’separator’/></Object>`. Separators are displayed as horizontal bars in the Administrator Interface menus, and cannot be selected.

CHAPTER 8

Private Labeling of Waveset

This chapter identifies the basic components you will need to rebrand the Waveset interface to match your company's intranet or corporate style guidelines. *Private labeling* is the customization of the interface to meet these corporate guidelines.

Private Labeling Tasks

There are several general categories of private labeling tasks:

- **Changing default header content by incorporating** your corporate logo, changing default text, and altering colors in both the User and Administrator interfaces.
- **Changing display fonts and component colors** throughout the application through the use of a style sheet located in `styles\customStyle.css`.
- **Changing default text** throughout the application by creating your own message catalog.
- **Changing Waveset behavior on commonly used pages** by editing the System Configuration object. These tasks, which include disabling the **Forgot Your Password?** button, are frequently performed by users while rebranding the product interface.

Architectural Features

Private labeling includes editing the components listed in the following table.

TABLE 8–1 Customizable Components

Component	Interface
`$WSHOME/styles/customStyle.css`	Administrator and User
`$WSHOME/WEB-INF/lib/idmcommon.jar`	Administrator and User
`$WSHOME/user/userFooter_beforeFirstTableRowTag.jsp`	User Interface

TABLE 8–1 Customizable Components *(Continued)*

Component	Interface
`$WSHOME/user/userFooter_beforeEndBodyTag.jsp`	User Interface
`$WSHOME/user/userFooter_beforeLastEndTableRowTag.jsp`	User Interface
`$WSHOME/includes/bodyEnd_beforeFirstTableRowTag.jsp`	Administrator Interface
`$WSHOME/includes/bodyEnd_beforeEndBodyTag.jsp`	Administrator Interface
`$WSHOME/includes/bodyEnd_beforeLastEndTableRowTag.jsp`	Administrator Interface
`$WSHOME/index_quickLinks.jsp`	Administrator and User

Style Sheets

Four *style sheets* affect the display characteristics of text in the product interface:

- `lockhart.css`. Contains Sun corporate interface styles for web applications.
- `style.css`. Defines the display attributes of pages throughout both interfaces. This file also controls the images contained in the headers.
- `customStyle.css`. Contains any changes to the default settings contained in `style.css`. and `lockhart.css`. Settings in this file override the settings in `style.css` and `lockhart.css`. Customers should not edit the preceding files, but instead put their customizations into `customStyle.css`.
- `Styles-Help.css`. Defines style classes used in online help and pop-up help (i-Help).

Default Text

Default text occurs throughout the product interface in the following:

- Form titles, subtitles, buttons, odd and even rows, section heads
- General text
- Warning messages
- Navigation button text, including both available and selected navigation buttons
- Table header/body text

Text Attributes

Display attributes include

- title: font-family, font-size, font-weight, color
- button: text-alignment, background-color

- text: same as title, text-decoration, white-space

Default Style Settings

The `$WSHOME/styles/style.css` and `lockhart.css` files contains default style settings. Do not edit these files.

Customized File

The `customStyle.css` file contains customizations and is not overwritten during product upgrades. Settings defined there will override the default settings in `style.css` and `lockhart.css`. *Include all your customizations in customStyle.css.*

JSP Files

Several JSP files contain the default settings for headers: `userHeader.jsp`, `bodyEnd.jsp`, and `bodyStart.jsp`. Do not edit these files. Instead, to preserve your customizations during product upgrade, edit only the JSPs listed in "Architectural Features" on page 127.

WPMessages_en.properties File

The `$WSHOME/WEB-INF/lib/idmcommon.jar` file contains the message catalog entries that you can extract into a `WPMessages_en.properties` file for editing.

Customizing Headers

Customization tasks are identical for both interfaces, although you must edit different files.

Note – Avoid editing any .jsp file other than the specified files. If you must edit one, first back up the .jsp to a safe location before copying, editing, and renaming it.

Changing Header Appearance

The most typical labeling tasks involve

- Changing the image referenced in the header section of the page from the default Sun logo to corporate standards. Replace or remove images by editing `customStyle.css`.

- Suppressing the Waveset logo.
- Using corporate internal look and feel guidelines, specifically borders, header, and background colors.
- Changing "logged in as ..." text. You cannot change this through .jsps. You can edit a custom message catalog. See "Changing Default Information Displayed in the Waveset User Interface Home Page" on page 133.

Customizing Waveset Pages

Typical customizations include:

- Creating a custom message catalog
- Customizing the home page
- Changing Default Strings in the Waveset User Interface Home Page

Creating a Custom Message Catalog

You must create a custom message catalog to override any string that is defined in the default message catalog. For example, if you want to rename Waveset, you must create a custom catalog and change the following definition:

```
PRODUCT_NAME=Waveset
```

The following entries also control the display of the product name:

```
LIGHTHOUSE_DISPLAY_NAME=[PRODUCT_NAME]
LIGHTHOUSE_TYPE_DISPLAY_NAME=[PRODUCT_NAME]
LIGHTHOUSE_DEFAULT_POLICY=Default [PRODUCT_NAME] Account Policy
```

See Chapter 9, "Customizing Message Catalogs," for detailed information about creating a message catalog.

Customizing the Home Page

- "Adding a List of Quick Links" on page 131
- "Changing the Default "Logged in as" Text" on page 131
- "Changing Page Title and Subtitle" on page 131
- "Changing Background Image on the Login Page" on page 132
- "Customizing the Browser Title Bar" on page 132

Adding a List of Quick Links

A typical customization to the home page involves adding a custom list of links to tasks or resources that users frequently access in your environment. These quick links offer a shortcut through the product interface to frequent destinations.

▼ To Add a Listof Quick Links

1. **Add links to the list in** `index_quickLinks.jsp`**.**
2. **Uncomment the list section by removing the surrounding <%-- and --%> tags.**
3. **Save the file. You do not need to restart your application server.**

Changing the Default "Logged in as" Text

▼ To Change the Default Text

1. **Imported the following XML file:**

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Configuration name='AltMsgCatalog'>
  <Extension>
    <CustomCatalog id='AltMsgCatalog' enabled='true'>
      <MessageSet language='en' country='US'>
        <Msg id='UI_BROWSER_TITLE_PROD_NAME_OVERRIDE'>Override Name</Msg>
      </MessageSet>
    </CustomCatalog>
  </Extension>
</Configuration>
```

2. **Add the following line to the System Configuration object within the** `<Configuration><Extension><Object>` **element:**

```
<Attribute name='customMessageCatalog' value='AltMsgCatalog'/>
```

3. **Add the following line:**

```
<msg id='UI_NAV_FOOT_LOG'>mytext{0}</msg>
```

4. **Save change and restart your application server.**

Changing Page Title and Subtitle

To change the default Login page title and subtitle and welcome message, change the following entries in the custom message catalog:

- `UI_LOGIN_TITLE`

- `UI_LOGIN_TITLE_TO_RESOURCE`
- `UI_LOGIN_WELCOME2`

To change this text, follow the procedure for extracting and editing the `WPMessages_en.properties` file detailed in "Changing the Default "Logged in as" Text" on page 131.

The following text lists the default message catalog settings:

```
UI_LOGIN_IN_PROGRESS_TITLE=Log In (In Progress)
UI_LOGIN_CHALLENGE=Enter Your {0} Password
UI_LOGIN_CHALLENGE_INFO=You are required to enter the password you logged into
[PRODUCT_NAME] with before the requested action can be completed.
UI_LOGIN_TITLE_LONG=[PRODUCT_NAME] LogIn
UI_LOGIN_WELCOME=Welcome to the Sun [PRODUCT_NAME] system. Enter the requested
information, and then click <b>Login</b>.
UI_LOGIN_WELCOME2=Welcome to the Sun [PRODUCT_NAME] system. Enter your user ID and
password, and then click <b>Login</b>. If you can't remember your password, click
<b>Forgot Your Password?</b>
UI_LOGIN_TITLE=Log In UI_LOGIN_TITLE_TO_RESOURCE=Log In to <b>{0}</b>
```

Changing Background Image on the Login Page

You can change the background image by editing `customStyle.css` as follows:

```
div#loginFormDiv {
   background:
url(.../images/other/login-backimage2.jpg)
no-repeat;
     margin-left: -10px;
     padding: 0px 0px 280px;
    height: 435px;
```

Customizing the Browser Title Bar

You can now replace the product name string in the browser title bar with a localizable string of your choice.

▼ To Replace the Product Name String

1 **Import the following XML file:**

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Configuration name='AltMsgCatalog'>
  <Extension>
    <CustomCatalog id='AltMsgCatalog' enabled='true'>
      <MessageSet language='en' country='US'>
        <Msg id='UI_BROWSER_TITLE_PROD_NAME_OVERRIDE'>Override Name</Msg>
      </MessageSet>
    </CustomCatalog>
  </Extension>
</Configuration>
```

2 **Using the Identity Manager IDE, load the System Configuration object for editing. Add the following line inside the `<Configuration><Extension><Object>` element:**

```
<Attribute name='customMessageCatalog' value='AltMsgCatalog'/>
```

3 **Also in the System Configuration object, you must change `ui.web.browserTitleProdNameOverride` to `true`.**

4 **Save this change to the System Configuration object, and restart your application server.**

Changing Default Information Displayed in the Waveset User Interface Home Page

The Waveset User Interface home page provides a "dashboard" area that summarizes basic information about the logged-in account, including the default strings listed in the following table.

All attributes belong to the ui.web.user.dashboard object.

TABLE 8–2 Default Settings of ui.web.user.dashboard Object

Default Configuration Object Setting	Description
`displayLoginFailures`	Displays the number of unsuccessful password or authentication question login attempts if a maximum login attempts value has been configured in an account's account policy.
`displayPasswordExpirationWarning`	Displays messages related to password expiration if password policy is applied to an account.

`displayApprovals`
`displayAttestationReviews`
`displayOtherWorkItems`
`displayRemediations`
	Enable the display of the following work item types for all users: Approvals, Attestations, Remediations, and Other.
	Note: Even if the configuration object is true for a particular type, the interface string may not appear based on the permissions granted to a user for his account.

`displayRequests`	Specifies the number of outstanding requests for role, group, or resource updates for an account
`displayDelegations`	Displays a string that indicates that the user has defined an approval delegation. Options include enabled or disabled.

By default, the value of the preceding configuration objects is set to true, and these strings will appear in the Waveset User Interface. To suppress the display of any string, set it to `false` in the System Configuration object.

You can edit the System Configuration object through the Waveset IDE. For general information about the System Configuration object, see Chapter 12, “Editing Configuration Objects”

Changing the Appearance of the User Interface Navigation Menus

In the Waveset User Interface, the horizontal navigation bar is driven by the End User Navigation User form in `enduser.xml`. The `userHeader.jsp`, which is included in all Waveset User Interface pages, includes another JSP named `menuStart.jsp`. This JSP accesses two system configuration objects:

- `ui.web.user.showMenu`. Toggles the display of the navigation menu on and off. The default value is `true`.
- `ui.web.user.menuLayout`. Determines whether the menu is rendered as a horizontal navigation bar with tabs (the default value is `horizontal`) or as a vertical tree menu (`vertical`).

The CSS style classes that determine how the menu is rendered are defined in `style.css`.

Note – If you implement custom JavaScript functions in the end user navigation bar (tabs), you must use endUserNavigation to reference that form (for example, `document.forms['endUserNavigation'].elements`).

Changing Font Characteristics

Display attributes typically specify the following basic font display characteristics.

TABLE 8–3 Font-related Display Characteristics

Display Attribute	Description
family	For example, Helvetica or Arial
size	Specified in point size (for example, 14 point)
weight	Unspecified indicates normal weight. When specified, typically bold
color	Typically specified as black (title -- font-family, font-size, font-weight, color

Certain components can be further defined by additional characteristics. For example, buttons can be defined with a background color. The alignment of the text and button label is another characteristic.

Editing Font Characteristics

To edit, copy from styles.css and paste into customStyle.css. Then, modify the selected setting in customStyle.css.

Example Entry

The following entry represents the default settings for each page title:

```
.title {
    font-family: Arial, Helvetica, sans-serif;
    font-size: 16pt;
    font-weight: bold;
    color: C;
```

Sample Labeling Exercises

The following example illustrates how to suppress the Waveset logo and reference a custom image in the masthead of the page. Consult the example files located in the sample/rebranding directory.

- Changing product name
- Changing masthead colors
- Changing navigation tab colors
- Changing Waveset behavior on commonly used pages

Replacing the Waveset Logo with a Custom Logo

To change the logo in the Administrator or User interfaces, copy the following snippets from styles/style.css into customStyle.css and replace the Waveset logo image with your .image file:

```
td.MstTdLogo {
    width: 31px;
    height: 55px;
    background: url(../images/other/logo.jpg) no-repeat 5px;
}

td.MstTdTtlProdNam {
    background: url(../images/other/xyz_masthead.jpg) no-repeat 10px 30px;
    padding:10px 10px 0px 10px;
    vertical-align:top;
    white-space:nowrap;
    height: 75px;
    width: 350px;
}
```

Note – For best results, create logo images between 50 and 60 pixels high.

Changing Masthead Appearance

▼ To Change the Appearance of Waveset

1. **Edit the `styles/customStyle.css` file.**

2. **Copy the following sections from `styles.css` into `customStyle.css` and modify as appropriate.**

```
MstDiv {background-image:url(../images/other/dot.gif);background-repeat:repeat-x;
background-position:left top;background-color:#000033}
.MstTblEnd {background: url(../images/other/dot.gif);background-color: #666;height: 1px;}
td.MstTblEndImg {
    background-color: #666;
    height: 1px;
    background: url(../images/other/dot.gif);
    font-size:1px;
}
td.MstTdLogo {
    width: 31px;
    height: 55px;
    background: url(../images/other/logo.jpg) no-repeat 5px;
}
td.MstTdSep {
    width: 1px;
    height: 61px;
    background: url(../images/other/dot.gif) no-repeat center;
}
td.MstTdTtlProdNam {
    background: url(../images/other/xyz_masthead.jpg) no-repeat 10px 30px;
    padding:10px 10px 0px 10px;
    vertical-align:top;
    white-space:nowrap;
    height: 75px;
    width: 350px;
}
```

Changing Navigation Tabs

This section describes how to customize the Waveset navigation bar.

Customizing the Waveset User Interface Navigation Bar

The Waveset User Interface implements a second XPRESS form that contains the navigation bar. This means that the rendered page has two `<FORM>` tags, each with a different `name` attribute:

```
<form name="endUserNavigation">
```

and

```
<form name="mainform">
```

When you insert JavaScript code into the Waveset User Interface navigation bar, be sure that you are referring the correct form. To do so, use the name attribute to specify which `<FORM>` tag you want to reference: `document.mainform` or `document.endUserNavigation`.

Customizing Navigation Links

Copy and edit the following code to customize the navigation links across the top of the page. Change the background-color to an appropriate color.

EXAMPLE 8–1 Customizing Navigation Links

```
* LEVEL 1 TABS */
.TabLvl1Div {
    background-image:url(../images/other/dot.gif);
    background-repeat:repeat-x;
    background-position:left bottom;
    background-color:#333366;
    padding:6px 10px 0px;
}
a.TabLvl1Lnk:link, a.TabLvl1Lnk:visited  {
    display:block;
    padding:4px 10px 3px;
    font: bold 0.95em sans-serif;
    color:#FFF;
    text-decoration:none;
    text-align:center;
}
table.TabLvl1Tbl td {
    background-image:url(../images/other/dot.gif);
    background-repeat:repeat-x;
    background-position:left top;
    background-color:#666699;
    border:solid 1px #aba1b5;
}
table.TabLvl1Tbl td.TabLvl1TblSelTd {
    background-color:#9999CC;
    background-image:url(../images/other/dot.gif);
    background-repeat:repeat-x;
    background-position:left bottom;
    border-bottom:none;
```

Changing Tab Panel Tabs

EXAMPLE 8–2 Changing Tab Panel Tabs Waveset

```
.
.TabLvl2Div {
    background-image:url(../images/other/dot.gif);
```

EXAMPLE 8–2 Changing Tab Panel Tabs Waveset *(Continued)*

```
    background-repeat:repeat-x;
    background-position:left bottom;
    background-color:#9999CC;
    padding:6px 0px 0px 10px
}
a.TabLvl2Lnk:link, a.TabLvl2Lnk:visited{
    display:block;
    padding:3px 6px 2px;
    font: 0.8em sans-serif;
    color:#333;
    text-decoration:none;
    text-align:center;
}
table.TabLvl2Tbl div.TabLvl2SelTxt {
    display:block;
    padding:3px 6px 2px;
    font: 0.8em sans-serif;
    color:#333;
    font-weight:normal;
    text-align:center;
}
table.TabLvl2Tbl td {
    background-image:url(../images/other/dot.gif);
    background-repeat:repeat-x;
    background-position:left top;
    background-color:#CCCCFF;
    border:solid 1px #aba1b5;
}
table.TabLvl2Tbl td.TabLvl2TblSelTd {
    border-bottom:none;
    background-image:url(../images/other/dot.gif);
    background-repeat:repeat-x;
    background-position:left bottom;
    background-color:#FFF;
    border-left:solid 1px #aba1b5;
    border-right:solid 1px #aba1b5;
    border-top:solid 1px #aba1b5;

table.Tab2TblNew td {
background-image:url(../images/other/dot.gif);
background-repeat:repeat-x;
background-position:left top;
background-color:#CCCCFF;
border:solid 1px #8f989f
}
table.Tab2TblNew td.Tab2TblSelTd {
border-bottom:none;
background-image:url(../images/other/dot.gif);
background-repeat:repeat-x;
background-position:left bottom;
background-color:#FFF;
border-left:solid 1px #8f989f;
border-right:solid 1px #8f989f;
border-top:solid 1px #8f989f
}
```

Changing Sorting Table Header

```
.tablehdr {
    background-image: url(../images/other/dot.gif);
    background-repeat: repeat-x;
    background-position: left bottom;
    background-color: #9999CC;
}
```

Changing User / Resource Table Component

EXAMPLE 8–3 Changing User / Resource Table Component

```
.tablesorthdr {
    /*background-color: #BEC7CC;*/
    background-image:url(../images/other/dot.gif);
    background-repeat:repeat-x;
    background-position:left bottom;
    background-color:#CCCCFF;
    border:solid 1px #aba1b5;
}

.treeContentLayout {
    background-color: #9999CC;
}

.treeBaseRow {
    padding-top: 0px;
    padding-left: 10px;
    padding-right: 10px;
    padding-bottom: 0px;
    background-color: #fff;
    border-left: solid 1px #8F989F;
    border-right: solid 1px #8F989F;
    border-bottom: solid 1px #8F989F;
}

.treeButtonCell {
    background-image:url(../images/other/dot.gif);
    background-color:#666699;
    color: #fff;
}

}

.treeMastHeadRow {
    background-color: #333366;
}

.treeMastHeadRow {
    background-color: #333366;
}

.treeMastHeadText {
    background-color: #333366;
```

EXAMPLE 8–3 Changing User / Resource Table Component *(Continued)*

```
    background-image: url(../images/other/dot.gif);
}
```

You can customize many other options (including text style and size, alignment, and the colors and configurations of other objects) by following the same procedures.

Note – To see the changes without rebooting your server or browser, perform a Ctrl-Refresh on a page.

Changing Waveset Behavior on Commonly Used Pages

You can customize many commonly altered properties of the User or Administrator interfaces can by editing the System Configuration object. The attribute `<Attribute name='ui'>` and its subobjects control the product interface. Modifying the attributes under this attribute can change the behavior of Waveset.

Miscellaneous Modifications: Admin Section of File

The admin section of System Configuration object file contains several attributes that are related to the Administrator Interface.

- To disable the **Forgot Your Password?** button on the Administrator login page, set `disableForgotPassword` to true.
- Setting `supressHostName` to true will suppress the display of the hostname for processes on the Task Details page.

EXAMPLE 8–4 Modifying the Admin Section of a File

```
<Attribute name='admin'>
   <Object>
      <Attribute name='disableForgotPassword'>
         <Boolean>false</Boolean>
      </Attribute>
      <Attribute name='workflowResults'>
         <Object>
            <Attribute name='suppressHostName'>
               <Boolean>false</Boolean>
            </Attribute>
         </Object>
      </Attribute>
   </Object>
</Attribute>
```

Miscellaneous Changes: User Section of the File

The user section of the System Configuration object file includes options for the User Interface.

Disable the **Forgot Your Password?** button by setting `disableForgotPassword` to `true`.

The `workflowResults` attribute contains attributes for customizing the display of workflows for non-administrative users, as indicated below.

TABLE 8–4 Attributes for Customizing Workflows for Non-Administrative Users

Attribute	Description
`anonSuppressReports`	Controls whether the workflow diagram is displayed in the anonymous user workflow status pages (`anonProcessStatus.jsp`).
`suppressHostName`	Controls whether the hostname is included in workflow status pages for end-users (`processStatus.jsp`).
`suppressReports`	Controls whether workflow diagrams is displayed to all non-anonymous users (`processStatus.jsp`).

EXAMPLE 8–5 Customizing Workflows for Non-Administrative Users

```
<Attribute name='user'>
   <Object>
      <Attribute name='disableForgotPassword'>
         <Boolean>false</Boolean>
      </Attribute>
      <Attribute name='workflowResults'>
         <Object>
            <Attribute name='anonSuppressReports'>
               <Boolean>false</Boolean>
            </Attribute>
            <Attribute name='suppressHostName'>
               <Boolean>false</Boolean>
            </Attribute>
            <Attribute name='suppressReports'>
               <Boolean>false</Boolean>
            </Attribute>
         </Object>
      </Attribute>
   </Object>
</Attribute>
```

To block the display of password and authentication question answers, set `obfuscateAnswers` to `true`. This setting causes answers to be displayed as asterisks in both the Administrator Interface and User Interface.

```
<Attribute name="obfuscateAnswers">
   <Boolean>true</Boolean>
</Attribute>
```

◆ ◆ ◆ CHAPTER 9

Customizing Message Catalogs

To add message catalog entries or modify entries provided with the system, you can create a customized message catalog.

Advantages of Custom Message Catalogs

Custom message catalogs provide the following benefits:

- Reduced maintenance in a clustered environment. Maintaining a separate message catalog means that you do not have to edit multiple copies of the `WPMessages.properties` file.
- Simplified version control. It is easier to track changes and back up revisions if the customized messages are located in one place.
- Upgrades to the product message catalog will not clash with any changes made to the customized entries.

How Waveset Retrieves Message Catalog Entries

Waveset retrieves message catalog entries in the following order:

- User-defined message catalog (Only one user-defined message catalog is permitted.)
- System-defined `defaultCustomCatalog` message catalog
- `config/WPMessages.properties` file
- `WPMessages.properties` file in the `idmcommon.jar` file

Message Catalog Format

In the `WPMessages.properties` file, entries are defined in the format *KeyName=MessageText*. In a customized message catalog, each entry is specified in a separate `Msg` element. The

KeyName is specified in the `id` attribute, while the *MessageText* is text between the `<Msg>` and `</Msg>` tags. The following example illustrates a message catalog entry:

```
<Msg id='UI_REMEMBER_PASSWORD'>Remember to set your password.</Msg>
```

The message text can contain HTML tags to control how the text is rendered, although this is not recommended. If you need to use HTML tags, use codes such as &lt; and &gt; instead symbols such as < and >.

Message text can also contain variables for data that Waveset will insert into the string when the string is displayed. The following example is the default message for the `AR_CORRELATED_USER` key:

```
Correlated account with user {0}.
```

The rendered version could appear as

```
Correlated account with user jdoe.
```

Creating a Customized Message Catalog

The following procedure describes how to create a user-defined message catalog.

▼ To Create a User-Defined Message Catalog

1 **If you are overriding default message catalog entries, locate the appropriate error message keys in the `WPMessages.properties` file. These keys must be specified in the customized message catalog.**

If you are creating new messages, confirm that the keys do not appear in the `WPMessages.properties` file

2 **Create an XML file or block with the following structure:**

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Configuration name='CatalogName'>
  <Extension>
    <CustomCatalog id='CatalogName' enabled='true'>
      <MessageSet language='en' country='US'>
        <Msg id='KeyName'>MessageText</Msg>
        <Msg id='KeyName'>MessageText</Msg>
        ...
      </MessageSet>
    </CustomCatalog>
  </Extension>
</Configuration>
```

where:

CatalogName is the name of the message catalog. This value will also be used to define the catalog in the System Configuration object.

KeyName is the message key name.

MessageText is a string that will be displayed on the graphical user interface. This text can contain HTML tags and variables.

If you are supporting a locale other than en_US, change the `language` and `country` attributes. If you are supporting multiple locales, create a separate `MessageSet` element for each locale.

See the Example section for a working sample.

3 **Import the file or block into Waveset.**

4 **Load the System Configuration object and add the following line within the** `<Configuration><Extension><Object>` **element:**

`<Attribute name='customMessageCatalog' value='CatalogName'/>`

5 **Save the changes to the System Configuration object.**

6 **Restart the application server. The new message catalog entries are now available to the system.**

Example Message Catalog

The following example creates a customized message catalog named `myCustomCatalog`. It replaces the label and help text for the **Import Exchange File** subtab.

EXAMPLE 9–1 Creating Customized `myCustomCatalog` Message Catalog

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Configuration PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Configuration name='myCustomCatalog'>
  <Extension>
    <CustomCatalog id='myCustomCatalog' enabled='true'>
      <MessageSet language='en' country='US'>
        <Msg id='UI_SUBNAV_CONFIGURE_IMPORT_EXCHANGE'>Import XML File</Msg>
        <Msg id='UI_SUBNAV_CONFIGURE_IMPORT_EXCHANGE_HELP'>Loads the specified XML file.</Msg>
      </MessageSet>
    </CustomCatalog>
  </Extension>
</Configuration>
```

CHAPTER 10

Developing Custom Adapters

Waveset'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 Waveset. These custom adapters define essential characteristics and methods needed to transform requests from Waveset into actions performed on a resource.

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

- "Before You Begin" on page 147
- "What is a Resource Adapter?" on page 149
- "Preparing for Adapter Development" on page 156
- "Writing Custom Adapters" on page 176
- "Installing Custom Adapters" on page 199
- "Testing Custom Adapters" on page 200
- "Troubleshooting Custom Adapters" on page 219
- "Maintaining Custom Adapters" on page 219

Note – Waveset 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" on page 148
- "Important Notes" on page 148
- "Related Documentation" on page 148

Intended Audience

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

- Developers who need to create custom resource adapters.
- Waveset administrators who are learning how the Waveset 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 Waveset resources and that you have read the Resources chapter of Business Administrator's Guide.

Important Notes

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

- 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 10–1 Related Documentation

Publication Title	Description
Resource Reference	Describes how to load and synchronize account information from a resource into Oracle Waveset.
Business Administrator's Guide	Contains additional information about customizing and managing resources supplied by Waveset.

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

What is a Resource Adapter?

A resource adapter serves as a proxy between Waveset 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 Waveset repository as a *resource object*. Waveset resource adapters are *standard* or *Active Sync-enabled* adapters.

This section contains the following topics:

- "What Are Standard Resource Adapters?" on page 149
- "What Are Active Sync-Enabled Resource Adapters?" on page 150
- "What is a Resource Object?" on page 155
- "What is a Resource Adapter Class?" on page 155

What Are Standard Resource Adapters?

Standard resource adapters provide a generic interface to resource types that are supported by Waveset; 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 Waveset 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 Waveset to the resource managed by Waveset:

1. Waveset 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 Waveset 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. Waveset 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. Waveset 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 using XML before writing the serialized object to the object repository.
4. Waveset displays the list of available resources in a multi-selection box when an Waveset user is created or modified.

   Selecting a resource causes Waveset to query the resource object for the available account attribute fields. Waveset 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 Waveset user object with the resource account information.

   When user account information is displayed, Waveset requests the list of resources on which the user has accounts from the saved account object. For each resource, Waveset 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 Waveset:

- 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 Waveset Active Directory resource to review this log every 30 minutes and trigger events in Waveset 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, Waveset 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 Waveset 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 Waveset. 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 Waveset 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 Waveset system used to update the identity. Typically, a workflow also handles errors within Waveset, often ending up as an Approval for a managing administrator.

7. Exceptions are logged in the Active Sync and Waveset 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 Waveset user or, if the adapter cannot match the user account, creates an Waveset user account.

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

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 Waveset identity and the resource account information (placed under the "account." namespace) are passed to the confirmation rule. The confirmation rule is then expected to return a value which may be expressed like a Boolean value. For example, "true" or "1" or "yes" and "false" or "0" or null. For the Database Table, Flat File, and PeopleSoft Component Active Sync adapters, the default confirmation rule is inherited from the reconciliation policy on the resource. The same confirmation rule can be used for reconciliation and Active Sync.

Parameter	Description
Correlation Rule	If no Waveset user's resource information is determined to own the resource account, the Correlation Rule is invoked to determine a list of potentially matching users/accountIDs or attribute conditions, used to match the user, based on the resource account attributes (in the account namespace). Returns one of the following types of information that can be used to correlate the entry with an existing Waveset account: ■ Waveset user name ■ `WSAttributes` object (used for attribute-based search) ■ List of `AttributeCondition` or `WSAttribute`-type items (`AND`-ed attribute-based search) ■ List of `String`-type items (each item is the Waveset ID or the user name of an Waveset account) If more than one Waveset account can be identified by the correlation rule, a confirmation rule or resolve process rule is required to handle the matches. For the Database Table, Flat File, and PeopleSoft Component Active Sync adapters, the default correlation rule is inherited from the reconciliation policy on the resource. The same correlation rule can be used for reconciliation and Active Sync.
Create Unmatched Accounts	If set to `true`, creates an account on the resource when no matching Waveset 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. 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`.

Parameter	Description
Process Rule	Either the name of a TaskDefinition or a rule that returns the name of a TaskDefinition, to run for every record in the feed. The Process rule gets the resource account attributes in the Active Sync namespace, as well as the resource ID and name. A Process rule controls all functionality that occurs when the system detects any change on the resource. It is used when full control of the account processing is required. As a result, a process rule overrides all other rules. If a Process rule is specified, the process will be run for every row regardless of any other settings on this adapter. At minimum, a process rule must perform the following functions: ■ Query for a matching User view. ■ If the User exists, checkout the view. If not, create the User. ■ Update or populate the view. ■ Checkin the User view. It is possible to synchronize objects other than User, such as LDAP Roles.
Resolve Process Rule	Name of the TaskDefinition or a rule that returns the name of a TaskDefinition to run in case of multiple matches to a record in the feed. The Resolve Process rule gets the resource account attributes as well as the resource ID and name. This rule is also needed if there were no matches and Create Unmatched Accounts was not selected. This workflow can be a process that prompts an administrator for manual action.

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 Waveset 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 Waveset, including the information described in the following table.

TABLE 10–2 Information Defined by Resource Objects

Type of Information	Sample Attributes
Connection information	■ Host name ■ Administrative account name ■ Administrative account Password
User attributes	■ First name ■ Last name ■ Phone numbers
Waveset attributes	■ List of approvers ■ Password policy for the resource ■ How many times to repeat attempts when contacting the resource

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

Note – You can view resource objects from Waveset's Debug pages:

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

Where:

- *host* is the local server on which Waveset 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 217 for more information.

What is a Resource Adapter Class?

A resource adapter class implements methods that

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

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 150 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" on page 156
- "Profile the Resource" on page 170
- "Decide Which Classes and Methods to Include" on page 173
- "Review the REF Kit" on page 173
- "Set Up the Build Environment" on page 174

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" on page 156
- "PrototypeXML String" on page 156
- "Resource Methods" on page 166
- "Considerations for Standard Resource Adapters" on page 167
- "Example Object Resource Attribute Declaration" on page 170

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 Waveset user interface. The `prototypeXML` string also defines resource objects stored in the Waveset repository.

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

Note – Some of these information types are specific to Active Sync-enabled adapters.

TABLE 10–3 `prototypeXML` Information Types

Type	Description
Resource	Defines top-level characteristics of the resource. Keywords include: ■ **syncSource**: If `true`, then adapter must be Active Sync-enabled. ■ **facets**: Specifies the modes enabled for this resource.
Resource attributes	XML elements that are defined with the `<ResourceAttribute>` element and used by Waveset to configure the resource. For more information, see "Resource Attributes" on page 157.
Account attributes	Defines the default schema map for basic user attributes. You use the `<AccountAttribute>` element to define account attributes. You map standard Waveset account attribute types differently than you map custom attributes. For more information about mapping account attributes to resource attributes, see "Map the Attributes" on page 179.
Identity template	Defines how the account name for the user is built. Use the `<Template>` tag to define this template. Account names are typically in one of the following forms: ■ An `accountId` is typically used for resources with a flat namespace such as Oracle. ■ 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. For more information, see "Identity Template" on page 164.
Login configuration	(*Standard resource adapter only*) Defines values to support pass-through authentication for the resource. Use the `<LoginConfigEntry>` element to define this value. For more information about pass-through authentication, see the Resource Reference.
Form	(*Active Sync-enabled adapters only*) Designates a form object that processes data from the Active Sync-enabled adapter before the data is integrated into Waveset. 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 Waveset 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. Waveset 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 Waveset Administrator interface, these attributes define the field names that are visible in the Waveset 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 Waveset administrator that the adapter will run as, scheduling and logging information, and additional attributes used only in Active Sync methods.

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

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 `&lt;` and `&apos;`.

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

TABLE 10–4 `<ResourceAttribute>` Element Keywords

Keyword	Description
`name`	Identifies the name of the attribute. **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.

TABLE 10–4 `<ResourceAttribute>` Element Keywords *(Continued)*

Keyword	Description
`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. Waveset 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 `&lt;` and `&gt;`. After the binary is translated, the `description` value looks like: `Description='<b>host</b> Enter the resource host name.'`
`facets`	Specifies the usage of this resource attribute. Valid values are ■ **provision**: Used in standard processing (default value). ■ **activesync**: Used in Active Sync processing for an Active Sync-enabled adapter.

The `ResourceAttribute>` element may also contain a `ValidationPolicy` element. A validation policy ensures the value a user specifies on the Resource Parameters page meets the requirements defined in a separate policy object.

The following sample causes the adapter to use the Port Policy to ensure the specified value is valid. The default Port Policy checks the value is an integer between 1 and 65536.

```
 <ResourceAttribute name='Port' value='123'>
    <ValidationPolicy>
      <ObjectRef type='Policy' id='#ID#PortPolicy' name='Port Policy'/>
    </ValidationPolicy>
  </ResourceAttribute>
```

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.

EXAMPLE 10–1 Overwriting the Resource Attribute Value for `template`

```
<Field name='template'>
   <Display class='Text'>
      <Property name='title' value='NDS User Template'/>
   </Display
</Field>
```

EXAMPLE 10–1 Overwriting the Resource Attribute Value for `template` *(Continued)*

```
<!-- Change NDS for the name of your NDS resource -->
<!-- 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 10–5 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 10–6 Active Sync-Specific Attributes Defined in `ACTIVE_SYNC_STD_RES_ATTRS_XML`

Required Resource Attribute	Description
`RA_PROXY_ADMINISTRATOR`	Waveset administrator for authorization and logging. This attribute corresponds to the Proxy Administrator field in the Waveset 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. ■ If you specify **`0`**, then a single log file is re-used. ■ If you specify **`-1`**, then log files are never discarded.

TABLE 10–6 Active Sync-Specific Attributes Defined in `ACTIVE_SYNC_STD_RES_ATTRS_XML` *(Continued)*

Required Resource Attribute	Description
`RA_MAX_AGE_LENGTH`	Specifies the maximum time before a log file is archived. ■ If you specify zero, then no time-based archival occurs. ■ 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 Waveset display.
`RA_LOG_PATH`	Absolute or relative path for the log file. This attribute corresponds to the Log File Path field in the Waveset display.
`RA_LOG_SIZE`	Maximum log file size. This attribute corresponds to the Maximum Log File Size field in the Waveset 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 10–7 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.

TABLE 10–7 Active Sync-Specific Attributes Defined in `ACTIVE_SYNC_EVENT_RES_ATTRS_XML` *(Continued)*

Required Resource Attribute	Description
`RA_CREATE_UNMATCHED`	■ If set to `true`, creates unmatched accounts. ■ 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`.

Waveset Account Attributes

Only available to Administrators defining the resource.

Waveset 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 Waveset user account. The Active Sync-enabled adapter collects these attributes and stores them in the global area for the input form.

Waveset 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 Waveset's performance.

You define Waveset 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 Waveset. For example, you would map the LDAP `sn` resource attribute to the `lastname` attribute in Waveset. Waveset account attributes include

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

Standard Adapter Schema Maps

You use the Account Attributes page, or schema map, to map Waveset 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 Waveset 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.

Waveset 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 Waveset 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 Waveset attributes to resource attributes.
- Create common Waveset attribute names to use with multiple resources.
- Identify required user attributes and attribute types.

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

For more information about creating a resource or editing a resource schema map, see the *Business Administrator's Guide*.

Identity Template

Note – An identity template is only available to Administrators who are defining the resource.

To view or edit the Waveset 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 Waveset 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.

Waveset 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 Waveset user object maintains a mapping between a user's identities and the resources to which they correspond.

The user has a primary `accountId` in Waveset 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 10–8 `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
- Oracle and Sybase relational databases

For resources with flat namespaces, the identity template can simply specify that the Waveset 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 10–9 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 Waveset 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 188 and the Resource Reference.

Resource Methods

Resource methods write information from Waveset 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 10–10 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 Waveset. 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 Waveset repository by performing the feed from the resource into Waveset.

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 Waveset 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 Waveset 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 10–11 `<AuthnProperty>` Element Attributes

Attribute	Description
`dataSource`	Specifies the source for the value of this property. Data sources for this property value include: ■ **user** (*Default*): Value provided by the user at login time. ■ **http attribute**: Value provided by the specified http session attribute. ■ **http header**: Value provided by the specified http header. ■ **http remote user**: Value provided by the http request's `remote user` property. ■ **http request**: Value provided by the specified http request parameter. ■ **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. ■ **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)
`isId`	Specifies whether this property value should be mapped to the Waveset `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:

▼ To Manage Users Across Forests

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 Resource Reference.

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

```
<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
                  Waveset. The entry defines two authentication properties whose dataSource value, if not
                  specified, is supplied by the user.
```

```
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.

```
" <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.

EXAMPLE 10–2 `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 Waveset 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" on page 171
- "Profiling an Active Sync-Enabled Resource Adapter" on page 171

Profiling a Standard Resource Adapter

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

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

  See Table 10–12 for a brief description of the default Waveset resource adapter files supplied with a standard Waveset 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 Waveset. 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 Waveset attributes.

  Decide which attributes you want to maintain in Waveset, determine what the attribute names are on the resource, and decide what to name the attributes in Waveset. 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 171:

- 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 Waveset 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 Waveset installations. Although some installations implement or are driven by external events, most Waveset 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 Waveset 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 Waveset to the resource or to create a feed from the resource to Waveset.

Review the REF Kit

The Sun Resource Extension Facility Kit (REF Kit) is supplied in the `/REF` directory on the Waveset 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 10–12 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: `/waveset/image/REF/javadoc/index.html`
`lib`	`REF/lib`	Jar files that you need to compile and test a custom adapter.

TABLE 10–12 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: ■ `MySQLResourceAdapter.java` for Database accounts ■ `ExampleTableResourceAdapter.java` for Database tables ■ `XMLResourceAdapter.java` for File-based accounts ■ `LDAPResourceAdapter.java` for simple methods when developing custom LDAP resource adapters ■ `LDAPResourceAdapterBase.java` for complex changes when developing custom LDAP resource adapters ■ `AIXResourceAdapter.java` for developing UNIX accounts ■ `SkeletonStandardResourceAdapter.java` for standard resources ■ `SkeletonStandardAndActiveSyncResourceAdapter.java` for standard and Active Sync-enabled resources ■ `SkeletonActiveSyncResourceAdapter.java` for Active Sync-only resources ■ `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 Waveset REF Kit.
`Waveset.properties`	`REF/config`	Copy of the properties file you need to test a custom adapter.

Set Up the Build Environment

This section contains instructions for setting up your build environment.

- "On Windows" on page 175
- "On UNIX" on page 175

Prerequisites:

You must install the JDK version required for your Waveset version. See “Supported Software and Environments” in the *Waveset 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:

▼ To Set Up Your Build Environment in Windows

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:

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

▼ To Set Up Your Build Environment in UNIX

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" on page 156 you are ready to start writing your custom adapter.

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

- "Process Overview" on page 176
- "Rename the Skeleton File" on page 178
- "Edit the Source File" on page 178
- "Map the Attributes" on page 179
- "Specify the Identity Template" on page 180
- "Write the Adapter Methods" on page 180
- "Configure the Adapter to Support Pass-Through Authentication" on page 192
- "Define the Resource Object Components" on page 193

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" on page 176
- "How To Write an Active Sync-Enabled Resource Adapter" on page 177

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.

▼ 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 178 for more information.**

3 **Edit the new adapter's source file as described in "Edit the Source File" on page 178.**

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 on the Microsoft Windows operating system.

▼ 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 178 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 178.**

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.

▼ To Edit the Source File

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 Waveset Administrator interface.

3. **Map the incoming resource attributes to Waveset 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 179 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 180 for more information.

5. **After editing the adapter file, you can load it into Waveset.**

Map the Attributes

Generally, you set options for the adapter type by mapping the incoming resource attributes to the standard Waveset 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 Waveset account attributes, use the syntax shown in the following example.

EXAMPLE 10–3 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 Waveset account attribute.
- The `<AttributeDefinitionRef>` element identifies the Waveset account attribute.

  The following table describes the `<AttributeDefinitionRef>` element fields.

TABLE 10–13 `<AttributeDefinitionRef>` Element Fields

Element Field	Description
`name`	Identifies the Waveset account attribute to which the resource attribute is being mapped. (The left column on the resource schema page in the Waveset 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 179.

Mapping Resource Attributes to Extended Schema Attributes

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

EXAMPLE 10–4 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 Waveset User interface pages:

- Resources
- Distinguished Name Template
- Edit Schema

For more information about identity templates, see "Identity Template" on page 164.

Write the Adapter Methods

The Waveset 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" on page 180
- "How to Write Active Sync-Enabled Adapter Methods" on page 189

How to Write Standard Resource Adapter-Specific Methods

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

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" on page 181
- "Connecting with the Resource" on page 181
- "Checking Connections and Operations" on page 182
- "Defining Features" on page 183
- "Disabling User Accounts" on page 187
- "Enabling Pass-Through Authentication for Resource Types" on page 188

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 Waveset, 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 10–14 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. 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 10–15 Methods Used to Check Communication

Method	Description
`checkCreateAccount`	Checks to see if an account can be created on the resource. The following features can be checked: ■ Can basic connectivity to the resource be established? ■ Does the account already exist? ■ Do the account attribute values comply with all (if any) resource-specific restrictions or policies that have not been checked at a higher level? This method does not check whether the account already exists. It contains the account attribute information needed to create the user account such as account name, password, and user name. After confirming that the account can be created, the method closes the connection to the resource.
`checkUpdateAccount`	Establishes a connection and checks to see if the account can be updated. This method receives a user object as input. It contains the account attribute information needed to create the user account such as account name, password, and user name. The user object specifies the account attributes that have been added or modified. Only these attributes are checked.
`checkDeleteAccount`	Checks to see if an account exists and can deleted. The following features can be checked: ■ Can basic connectivity to the resource be established? ■ Does the account already exist? ■ Do the account attribute values comply with all (if any) resource-specific restrictions or policies that haven't been checked at a higher level? This method does not check whether the account already exists. It receives a user object as input. It contains the account attribute information needed to delete the user account such as account name, password, and user name. After checking to see if the account can be deleted, the method closes the connection to the resource

Defining Features

The `getFeatures()` method specifies which features are supported by an adapter. The features can be categorized as follows:

- General features
- Account features
- Group features
- Organizational unit features

The `ResourceAdapterBase` class defines a base implementation of the `getFeatures()` method. The Enabled in Base? column in the following tables indicates whether the feature is defined as enabled in the base implementation in `ResourceAdapterBase`.

TABLE 10–16 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 10–17 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.
`ACCOUNT_EXCLUDE`	No	Determines whether administrative accounts can be excluded from Waveset. Override the `supportsExcludedAccounts` method with a `true` value to enable this feature.

TABLE 10–17 Account Features *(Continued)*

Feature Name	Enabled in Base?	Comments
`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 Waveset 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 10–18 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 10–19 Organizational Unit Features

Feature Name	Enabled in Base?	Comments
`ORGUNIT_CREATE` `ORGUNIT_DELETEORG` `UNIT_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 10–20 Creating Accounts on the Resource

Method	Description
`realCreate()`	Creates the account on the resource. Receives a user object as input, and contains the account attribute information needed to create th user account (such as account name, password, and user name)

TABLE 10–21 Deleting Accounts on the Resource

Method	Description
`realDelete()`	Deletes one or more accounts on the resource. Receives a user object or list of user objects as input. By default, this method creates a connection, `realDelete`, closes the connection for each user object in the list.

TABLE 10–22 Updating Accounts on the Resource

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

TABLE 10–23 Getting User Information

Method	Description
`getUser()`	Retrieves information from the resource about user attributes. Receives a user object as input (typically with only an account identity set), and returns a new user object with values set for any attribute defined in resource schema map.

You can use list methods to establish processes that adapters use to retrieve user information from the resource.

TABLE 10–24 List Methods

Method	Description
`getAccountIterator()`	Used to discover or import all the users from a resource. Implements the Account Iterator interface to iterate over all users of a resource.
`listAllObjects ()`	Given a resource object type (such as `accountID` or group), returns a list of that type from the resource. Implement this method to generate lists that are used by the resource, such as a list of resource groups or distribution lists. This method is called from the user form (not called by provisioning engine).

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 Waveset can work with.

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 10–25 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 Waveset.

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.
- **Waveset disable utility**: If the resource does not support disabling an account, or supports disable by means of resetting the user's password, the Waveset 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 Waveset 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:

▼ To Enable Pass-Through Authentication on 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 Waveset.

```
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 Waveset, 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" on page 189
- "Polling the Resource" on page 190
- "Storing and Retrieving Adapter Attributes" on page 191
- "Updating the Waveset Repository" on page 191
- "Shutting Down the Adapter" on page 191

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 Waveset 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 10–6 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 10–26 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.

EXAMPLE 10–5 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 Waveset 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 Waveset object
- Update that object with the changed information

Mapping the Changes to the Waveset 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 Deployment Reference.

In `SkeletonActiveSyncResourceAdapter`, this process is handled in the `buildEvent` and `processUpdates` methods.

Shutting Down the Adapter

No system requirements are associated with adapter shutdown. Waveset 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

Waveset uses pass-through authentication to grant users and administrators access through one or more different passwords. Waveset 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" on page 193
- "Defining Resource ObjectTypes" on page 195
- "Defining Resource Object Features" on page 196
- "Defining Resource Object Attributes" on page 197
- "Defining Resource Forms" on page 198

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 Waveset, 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, Waveset performs an AND operation on these object classes prior to any list or get requests for this object type.

EXAMPLE 10–6 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`.

EXAMPLE 10–7 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`.

EXAMPLE 10–8 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 Waveset, 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 Waveset for editing and searching

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

TABLE 10–27 Supported `<ObjectType>` Element Attributes

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

The following example shows `ObjectType` definitions:

EXAMPLE 10–9 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, Waveset 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 10–28 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 Waveset.

EXAMPLE 10–10 `<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 Waveset. Each <ObjectAttribute> element name should be the same as the native resource attribute name. Unlike user attributes in Waveset, no attribute mapping is specified. Use only the native attribute names.

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

TABLE 10–29 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 Waveset (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>.

EXAMPLE 10–11 <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 10–30 <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 Waveset

---

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 Waveset. 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 10–31 Top-Level Namespace Attributes

Attribute	Description
`<objectType>.resourceType`	Waveset resource type name (for example, LDAP, Active Directory)
`<objectType>.resourceName`	Waveset resource name
`<objectType>.resourceId`	Waveset 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`	Waveset member organization
`<objectType>.attrsToGet`	List of object type specific attributes to return when requesting an object through `checkoutView` or `getView` (list)

TABLE 10–31 Top-Level Namespace Attributes *(Continued)*

Attribute	Description
`<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. Waveset 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 Installation Guide.**

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 Deployment Reference 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 Waveset using your adapter.**

9 **Ensure your native managed system is operational.**

10 **Test the connectivity of your new Waveset resource, as described in "Checking Connections and Operations" on page 182.**

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" on page 200
- "Compatibility Testing Your Adapter" on page 201

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):

▼ To Unit Test Your Adapter

1 **Save the adapter.**

2 **Run unit tests on that adapter from your own machine.**

3 **Load the adapter into Waveset.**

4 **Test the adapter in Waveset, as follows:**

a. **Log into the Waveset 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 Waveset. Even well-written resource adapters will sometimes not work well after the external resource has been upgraded.

Waveset 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 Waveset's Compatibility Test Suite. The information is organized as follows:

- "How the Compatibility Test Suite Works" on page 201
- "How to Run the Compatibility Tests" on page 202
- "Example 1: Using the Default DataProvider to Run Compatibility Tests" on page 203
- "Example 2: Adding More Data" on page 205
- "Example 3: Finishing the Test Configuration" on page 207
- "Example 4: Executing Javascript or Beanshell Script" on page 210
- "Example 5: Running Tests from Inside the Web Container" on page 213

How the Compatibility Test Suite Works

Waveset'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, Waveset 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 Waveset) 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 Waveset Compatibility Test Suite

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:`

Argument	Description
`-propsFile` *value*	Path to the properties file
`-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 the user on
`-import` *value*	Comma separated list of files (on server) to import
`-toDir` *value*	Directory to put test output in
`-v`	Echoes all arguments passed in to the screen
`-h`	Displays the 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 **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 Waveset 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.

Prepare the Test

Use the following steps to prepare the compatibility tests.

▼ To Prepare the 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.**

Execute the Test

Use the following steps to execute the compatibility tests.

▼ To Execute the Compatibility Test

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:

```
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 “Execute the Test” on page 204, 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.

“Execute the Test” on page 204 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, Waveset 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 “Execute the Test” on page 204, 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.

Prepare the Test

Use the following steps to prepare the compatibility tests.

▼ To Prepare the 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.**

Execute the Test

Use the following steps to execute the compatibility tests.

▼ To Execute the Test

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:

```
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.

Prepare the Test

Use the following steps to prepare the compatibility tests.

▼ To Prepare the Test

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.**

Execute the Tests

Use the following steps to execute the compatibility tests.

▼ To Execute the Tests

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:

```
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.

Waveset 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.

Prepare the Test

Prepare the compatibility test to execute with a beanshell script

▼ To Prepare the Test

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.**

Execute the Tests

Execute the compatibility tests as follows:

▼ To Execute the Test

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:

```
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/SimulatedCompatibilityConfig.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.

Prepare the Test

Use the following steps to prepare the compatibility tests.

▼ To Prepare the Test

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 **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 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

Property	Description
`importScriptText`	String containing script to run
`importScriptSuffix`	Specify `bsh` if the script is a beanshell script Specify `js` if the script is javascript. **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.**

Execute the Tests

Run the tests from the `CompatTests.jsp` page

To Execute the Test

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 e ... -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 (`\`).

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
```

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
```

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
```

Running a Compatability 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" on page 217
- "Testing the Resource Object in Waveset" on page 218

Viewing and Editing a Resource Object

You can confirm the configuration of your resource by viewing the raw XML in the repository.

▼ To View and Edit a Resource Object

1. **Log into the Administrator user interface.**

2. **Open the Waveset 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 Waveset 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 Waveset

You can use the Find Resources and List Resources pages in the Waveset Administrative interface to test your implementation of a resource object.

Select Resource > List Resource to confirm the following performance characteristics.

TABLE 10–32 List Resource Performance Characteristics

Expected Behavior in Interface	If This Does Not Occur
Waveset includes your resource type in the drop-down list of possible new resources.	Confirm that you have added it to the `resource.adapters` attribute in the `Waveset.properties` file.
When you open the resource folder, its contents reflect all the `<ObjectType>` elements that are defined in your resource adapter's `<ObjectTypes>` section.	Review the `<ObjectType>` elements in the adapter's `prototypeXML`.
When you right-click on one of your resource object types, all the supported features specified in your resource adapter's `<ObjectFeatures>` section per `<ObjectType>` is available from the menu.	Go to the Debug page and view or edit the resource in question to ensure that its list of `<ObjectFeatures>` for the `<ObjectType>` in question is correct.
You can create new resources and update existing resource objects.	Verify that your resource adapter code is in `WEB-INF/classes/com/waveset/adapter/sample`
The correct ResourceForms have been loaded for each type of operation.	■ Confirm that you have checked in all needed resource forms ■ Verify that the forms are correctly referenced (including the correct case) in the section for forms in the System configuration object.

Select Resource > Find Resources to confirm the following performance characteristics.

TABLE 10–33 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 Waveset 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 System Administrator's Guide for detailed information about tracing and debugging custom adapters.

Maintaining Custom Adapters

Any time you install a new Waveset 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 Waveset version. All custom Java that uses Waveset APIs (including custom resource adapters) require a recompile during upgrading. Also, consider other Java classes that use the Waveset library.

For more information about upgrading, see Upgrade Guide.

Note – If your current Waveset installation has a large amount of custom work, contact your Sun Account Representative or Sun Customer Support for assistance with your upgrade.

CHAPTER 11

Optimistic Checkouts

By default, Waveset workflow locks User objects pessimistically. This means that Waveset assumes that any concurrent change to the same User object is likely to conflict. Therefore, to prevent overwriting (and losing) previous changes, Waveset first locks the User object when the User View is checked out thus preventing concurrent edits to the same user. It then also ensures that only one Update User workflow is allowed to launch for a given user when the User View is checked in. Taken together, these two mechanisms define the current pessimistic approach to updating users in Waveset.

This pessimistic assumption is the safest course, but this behavior limits the throughput of change operations for a particular User object. It would be desirable for Waveset to allow non-overlapping sets of changes to proceed independently. For example, it might be preferable to allow an ActiveSync process to change resource account information for a User object while another workflow that changes the roles for the same User object is still going through approvals.

A common use case is the scenario where changes to each User are segmented by target resource and are thus largely independent of one another. A change to one resource account for a User object should not block a change to another resource account for the same User.

Optimistic checkout addresses these issues by allowing checkout and checkin of the User view concurrently without locking the user repository object at view checkout time. It also allows multiple Update User workflows to run concurrently. At the time User account reprovisioning occurs, any changes made in the currently executing workflow are checked against “remote” changes that have occurred to the User view since the time the view was checked out. If conflicts are detected, an exception is thrown. If no conflicts are detected, the local changes are merged with the remote changes and committed.

Principles of Optimistic Checkouts

The following principles apply to optimistic checkout:

- **Optimistic checkout enables concurrent changes to the same User view .** Any number of callers (such as workflows) could checkout the same User view optimistically. Each caller could concurrently modify the same User view. Each caller would succeed in reprovisioning the user as long as the caller's desired changes do not conflict with intervening changes made by any and all other callers.
- **Optimistic checkout does not prevent pessimistic checkout.** An optimistic checkout of a User view is not a shared lock. Specifically, an optimistic checkout of a User view would not prevent another caller from checking out the same User view pessimistically.
- **Pessimistic checkout takes precedence over optimistic checkout.** A pessimistic checkout of a User view would still prevent any other user from checking out the same User view pessimistically. The existence of a pessimistic update user task would also prevent another any other caller from reprovisioning the same user since the pessimistic name-based task workflow lock remains in effect.

  However, once the caller who makes a pessimistic checkout of a User view checks in the same User view or otherwise abandons that caller's lock on the User repository object and that task terminates, any other caller who checked out the same User view optimistically may succeed in reprovisioning the user as long as the second caller's changes do not conflict with the changes that the first caller made.

  Optimistic tasks can launch but wait (with configurable retry interval and count) until a running pessimistic task completes before computing conflicts and attempting reprovisioning. Only one pessimistic task may launch at one time as guaranteed by the existing task name based workflow lock.
- **Checkouts are pessimistic by default.** Pessimistic checkout remains the default behavior of the Waveset product for any installation of Waveset that is upgraded to a release that supports optimistic locking. This preserves backward-compatibility for older deployments.
- **Waveset detects conflicts.** A conflict is defined as an overlap between two sets of changes produced by the existing UpdateViewer difference calculation. For example, suppose that a caller calls `checkoutView("User:joebob")` optimistically. Our caller locally modifies `waveset.email`, and then calls `checkinView()`. If no other caller has modified `waveset.email` in that same User view since the point in time that our caller made his optimistic checkout, then our caller's `checkinView()` should succeed.

  However, suppose that a caller changes a field within an existing account for an Exchange Server resource. Within the User view, the caller modifies `accounts[ExchangeServer].Profile`. As long as no other caller has modified that same field (since the point in time that our caller made his optimistic checkout), then the caller's `checkinView()` should succeed.

- **The system merges changes.** If no conflicts are detected, then the changes made to the user are merged into the current User View. In this way, both our caller's changes and intervening changes to the User View are preserved.
- **Optimistic reprovisioning attempts can be retried.** At the point in the Update User workflow where reprovisioning the User object begins, if optimistic checkout is enabled, the workflow attempts to lock the User repository object. It also tries to ensure that no pessimistic Update User workflows are executing concurrently. If either of these conditions cannot be met then, by default, the optimistic workflow sleeps briefly then retries. Both the retry interval and retry count values are configurable view options.
- **Reprovisioning can be forced.** In the case that conflicts are detected, an error result is returned from the call to `WorkflowServices#reProvision()`. The list of conflicts is returned in a format similar to the change objects currently calculated by the UpdateViewer and includes the local new value, remote new value, and the original value for each conflict. The default Update User workflow detects this condition and terminates, thereby abandoning the changes.

  You can customize a workflow to allow for some type of remediation. A caller may wish to force the changes to be committed after the remediation process. Setting the view option OP_IGNORE_CONFLICTS to "true" at view checkin time enables this behavior. If it is set and evaluates to boolean true when `WorkflowServices.reProvision()` is called, then conflict checking is performed. In addition, Waveset merges the data, despite any detected conflicts. However, during the merge process where conflicts do occur (either in single-valued attributes or for individual elements in lists of named GenericObjects), the local change will be committed, thus overwriting any such intervening changes. Non-conflicting intervening changes are preserved. The caller assumes complete responsibility for the possibility of overwriting such conflicting intervening changes. However it should be noted that conflicts are specific to a particular point in time. Once an initial call to `reProvision()` returns a conflict error and releases the user object lock, other tasks may make intervening changes before the second call to force reprovisioning can occur.

Detecting Conflicts

Conflict detection is based on the existing UpdateViewer change detection code. This code collects changes to the user view for each account assigned to the user. It also detects changes to a set of hardcoded attributes in the waveset view and also to extended attributes. The list of hardcoded updatable waveset attributes includes:

```
Constants.ACCOUNT_ID
Constants.PASSWORD
Constants.EMAIL
"disabled"
"locked"
"lockExpiry"
"questionLocked"
```

```
"questionLockExpiry"
"correlationKey"
"passwordExpiry"
"passwordExpiryWarning"
"organization"
"roleInfos"
"applications"
"resources"
"resourceAssignments"
"exclusions"
"assignedLhPolicy"
"adminRoles"
"capabilities"
"controlledOrganizations"
"userForm"
"viewUserForm"
"customForm"
"forwardingApprover"
"questions"
"suppliedQuestions"
"idmManager"
"delegates"
```

In the Administrator interface, such changes to these User view attributes are detected by the existing UpdateViewer and presented to the administrator for confirmation when the Save button is pressed. It appears as a list of attribute names with corresponding old and new values.

Conflict detection for optimistic checkout is based on the UpdateViewer update view and proceeds as follows:

1. The list of local changes is fetched from the view. The UpdateViewer has already calculated this change list, and it is present in the User view.
2. The list of remote changes is computed by fetching the baseline view stored in the view at checkout time. These changes are compared to a fresh view based on the current repository User object in the same manner as the local changes are computed. A similar change list is produced.
3. If a changed attribute appears in either the local or remote changes but not both, then no conflict is indicated.
4. If a changed attribute appears in both the local and remote changes and has the same new value in both, then no conflict is indicated. This comparison is straightforward for attributes whose values are not lists. However, attributes that contain lists are processed differently, depending on the type of list.

The following types of lists need to be considered when checking for conflicts:

- "Non-Named GenericObject Lists" on page 225
- "Lists of Named GenericObject Elements" on page 225

Non-Named GenericObject Lists

Lists composed of elements which are not named GenericObjects never produce a conflict. Since element-level list conflict detection is predicated on the ability to identify list elements (by name) and no such name exists in this case, no conflicts can occur - list order is not significant. As such, for such lists, conflict detection is bypassed. Merging, however, still occurs at the element level.

For example, consider an attribute whose value is a list of strings and whose initial value is ["A", "B", "C"]. Then two edits (such as two UserView `checkout()` methods) occur with this as the starting state for this attribute value. The first edit removes element B, resulting in a final list value of ["A", "C"] and commits that change. The second edit (an optimistic edit) starts with the same initial value of the list (["A", "B", "C"] but then removes element "A" and adds element "D" giving a local end result attribute value of ["B", "C", "D"]. The final merged list value is ["C", "D"] because:

- The first edit removed "B".
- The second edit removed "A".
- The second edit added "D".

Lists of Named GenericObject Elements

Assuming named GenericObjects with the same name represent the same object allows for better and more granular conflict detection. Changes in a named element are identifiable since list elements with the same name are assumed to be the same object. In this case, a failing equality comparison indicates a conflict. List additions and deletions are also similarly discernible.

For lists of named GenericObjects, Waveset can detect whether an element has been added, deleted or modified in both the local and remote changes. The system then compares the resulting two lists to determine if a conflict is indicated using the following rules:

TABLE 11–1 Local and Remote Conflict Checks

	Local Add	Local Delete	Local Change
Remote Add	Add conflict is not equal		
Remote Delete		No	Yes
Remote Change		Yes	Conflict if not equal

The blank cells in the table indicate that such a case is not possible. Since the baseline view is the same for computing both local and remote changes, a list addition in one set of changes indicates that the value was not present in the baseline. Therefore, it is not possible to delete or change this (non-existent) element value in the local or remote changes. Similarly, the deletion of an element or change to an element value indicates that the value was present, and an addition of that value is not possible.

An example of such a list that contains named GenericObject elements is the roleinfos attribute. Both the old and new values are lists of objects which are named by the role name. An addition or deletion of such a named element can be determined by querying the list by object name. For cases where the named element is present in the list in both the local and remote changes, the (Generic) objects can be compared for equality. If the new values are identical (equal), then this is not considered a conflict.

Implementing Optimistic Checkouts

The section provides information about the Waveset objects that are used to implement optimistic checkouts.

- "User View" on page 226
- "Conflict Results Objects" on page 227

User View

The UserViewConstants.java file defines the following options to support optimistic checkouts. These can be set as view options when the User View is checked out in the same manner as existing view options.

- OP_OPTIMISTIC - If this option is set to True, optimistic checkout semantics are enabled. This option must be set at view checkout time and at view checkin time so that a baseline copy of the view can be taken and optimistic reprovisioning can occur. This baseline copy of the view is used during the subsequent view checkin to determine any intervening changes that may have occurred to the same User by other Update User workflow reprovisioning operations. These intervening changes, if any, are compared against the local changes to determine if any conflicts are present and, if not, to merge the local changes with the intervening remote changes to complete the reprovisioning process for that User.
- OP_OPTIMISTIC_RETRY_COUNT - The number of times a caller can to retry a reprovisioning operation. The default value is 3 times.

  The reprovisioning operation attempts to lock the User repository object and ensures that no pessimistic Update User workflow is running for the same User before performing optimistic reprovisioning. If either of these conditions are not met, the task retries after sleeping for the specified interval. If the retry count is exceeded, an exception is thrown.
- OP_OPTIMISTIC_RETRY_INTERVAL - The number of milliseconds to wait between reprovisioning retries. The default value is 30,000 milliseconds (30 seconds).
- OP_IGNORE_CONFLICTS - This view option can be passed at user view checkin time to force the local changes to be committed. If it is set and evaluates to True, then the reprovisioning behaves in an optimistic manner with the exception that conflicts, if found, are not considered an error. Merging is still performed with local change values being committed in cases of conflict, thus overwriting these conflicting intervening changes. Non-conflicting changes are preserved. In the case where no other errors occurred but

conflicts were found and OP_IGNORE_CONFLICTS is set to true, the WavesetResult will contain a named result (called conflicts) but will not have an error result set.

The following example enables optimistic checkout in a JSP (such as account/modify.jsp):

```
try {
    String id = requestState.getParameter(requestState, "id");
     if (id == null) {
         form.setTitle(Messages.UI_ACCT_CRT_USR_TITLE);
         form.setViewId("User");
     }
     else {
         form.setTitle(Messages.UI_ACCT_EDIT_USR_TITLE);
         form.setViewId("User:" + id);
            // Set this option so that checkin of the view
           // won't launch a reprovision.  This does however mean that the user
           // changes will be stored.  Need to be able to defer this.
           //     requestState.setOption(UserViewConstants.OP_NO_REPROVISION, "true");

            requestState.setOption(UserViewConstants.OP_SELECT_RESOURCES, "true");

           // Enable Optimistic Checkout
           requestState.setOption(UserViewConstants.OP_OPTIMISTIC, "true");

          // Set this options to true to build the
          // "Forwarding Approvers" select list
          req.setOption(UserViewConstants.OP_BUILD_APPROVER_LISTS, "true");
     }
```

Conflict Results Objects

Conflicts, if any, are returned in the WavesetResult returned by `WorkflowServices#reProvision()` in a named GenericObject called conflicts. This is normally further returned in the GenericObject result from the call to view checkin. This object contains an attribute named `accounts` which is a list, by account, of attribute conflicts for that account.

For each such conflict detected, a GenericObject representing the conflict is created whose name is the name of the attribute.

The following example shows a set of conflicts.

```
<Object>
   <Attribute name='accounts'>
     <List>
       <Object name='SimRes1'>
         <Attribute name='conflicts'>
           <List>
             <Object name='attr1'>
               <Attribute name='local' value='Safari Attr1'/>
               <Attribute name='original' value='Orig Attr1'/>
               <Attribute name='remote' value='Firefox Attr1'/>
             </Object>
             <Object name='idmManager'>
```

```
        <Attribute name='local' value='Mr. Safari'/>
        <Attribute name='original' value='Mr. Orig'/>
        <Attribute name='remote' value='Mr. Firefox'/>
      </Object>
      <Object name='email'>
        <Attribute name='local' value='safari_email'/>
        <Attribute name='original' value='orig_email'/>
        <Attribute name='remote' value='firefox_email'/>
      </Object>
    </List>
  </Attribute>
</Object>
<Object name='Lighthouse'>
  <Attribute name='conflicts'>
    <List>
      <Object name='idmManager'>
        <Attribute name='local' value='Mr. Safari'/>
        <Attribute name='original' value='Mr. Orig'/>
        <Attribute name='remote' value='Mr. Firefox'/>
      </Object>
      <Object name='email'>
        <Attribute name='local' value='safari_email'/>
        <Attribute name='original' value='orig_email'/>
        <Attribute name='remote' value='firefox_email'/>
      </Object>
      <Object>
        <Attribute name='roleInfos'>
          <List>
            <Object name='IT Role1'>
              <Attribute name='local'>
                <GenericAttribute>
                  <Object>
                    <Attribute name='attribute'>
                      <Object name='IT Role1'>
                        <Attribute name='assignedBy'>
                          <List>
                            <String>Business Role 2</String>
                          </List>
                        </Attribute>
                        <Attribute name='assignmentType' value='required'/>
                        <Attribute name='state' value='assigned'/>
                        <Attribute name='type' value='ITRole'/>
                      </Object>
                    </Attribute>
                  </Object>
                </GenericAttribute>
              </Attribute>
              <Attribute name='original'/>
              <Attribute name='remote'>
                <GenericAttribute>
                  <Object>
                    <Attribute name='attribute'>
                      <Object name='IT Role1'>
                        <Attribute name='assignedBy'>
                          <List>
                            <String>BusinessRole1</String>
                          </List>
                        </Attribute>
                        <Attribute name='assignmentType' value='required'/>
```

```
                              <Attribute name='state' value='assigned'/>
                              <Attribute name='type' value='ITRole'/>
                            </Object>
                          </Attribute>
                        </Object>
                      </GenericAttribute>
                    </Attribute>
                  </Object>
                </List>
              </Attribute>
            </Object>
          </List>
        </Attribute>
      </Object>
    </List>
  </Attribute>
</Object>
```

In this example, conflicts have been detected in two accounts: SimRes1 and Lighthouse. The attributes on SimRes1 for which conflicts have been detected are `attr1`, `idmManager`, and `email`. For Lighthouse, the attributes in conflict are `idmManager`, `email`, and `roleInfos`. For each conflict the original, remote, and local changed values are listed. The original value is the value of the attribute at the time the view was checked out. The remote value is the value of the attribute present in the repository (or on the resource) at the time reprovisioning started. The local value is the changed value of the attribute made by the currently executing task.

The `idmManager` attribute had a value of `Mr. Orig` when the currently running task checked out the User view (the original attribute value). The current task changed that value to `Mr. Safari` (the local attribute value). However, in the interim, one or more other changes have been committed which changed the attribute value to `Mr. Firefox` (the remote attribute value). This constitutes a conflict because the remote changes (`Mr. Safari`) would be lost if the local changes were committed.

CHAPTER 12

Editing Configuration Objects

This chapter introduces an Waveset component called a *configuration object*. Configuration objects store persistent customizations to Waveset. They are cached object types, which means that all configuration objects are brought into memory, and the cache is subsequently flushed, whenever a configuration object is changed. Speaking broadly, with the large exception of `User` objects and `TaskInstances`, most objects in the Waveset repository are configuration objects.

Editing configuration object properties is one way of implementing persistent changes to Waveset behavior. This appendix describes how to view and edit configuration objects. The information is organized as follows:

- "Data Storage" on page 231
- "Viewing and Editing Configuration Objects" on page 233
- "Refreshing User Objects" on page 245

Data Storage

Oracle Waveset repository stores configuration object data in the following tables:

Note – This is not a comprehensive list of tables; only relevant tables are listed here.

- **object**. Stores most of the Waveset application's configuration objects, which include:
  - `SystemConfiguration` objects
  - `TaskDefinition` objects
  - `WorkItem` objects
  - `Form` objects
  - `Resource` objects
- **task**. Stores `TaskInstances` and `WorkItems` (in other words, the nonstatic instances of Workflows).
- **org**. Stores all Waveset organizational information.

- **userobj**. Stores all the Waveset enterprise identities. These identities are simply containers for accounts on managed systems.
- **account**. Stores the accounts identified on a managed system.

> **Note –** This is the table being referenced when someone "builds the account index." Each managed system has a set number of accounts in this table after reconciliation or account linking.

- **log**. Stores all audit events and log type information.

A key concept to understand in the Waveset repository is that all data is stored in two ways, where each table has indexed and keyed columns used to query objects and each table has an XML column used to store the entire ASCII representation of the object (depending on the database engine this is typically a `BLOB` or `MEDIUM TEXT` data type). Waveset stores data in this way because all Waveset objects are de-serialized from Java objects to ASCII XML for storage in the repository.

The application, at a high level, queries by the indexed columns, pulls back XML ASCII text and then serializes the XML into Java objects. These objects are usually made available through the use of views (such as User view and Password view).

Object Naming Conventions

Do not use the following characters in any Waveset object names:

Character	Description
’	single quotation mark
=	equal sign
.	period
|	vertical bar
[	left bracket
]	right bright
,	comma
:	colon
$	dollar sign
\	backslash

Character	Description
"	double quotation mark

Avoid using other special characters in object names, such as the following, to prevent potential errors:

Character	Description
_	underscore
%	percent sign
*	asterisk
#	number sign
^	caret

Viewing and Editing Configuration Objects

Editing configuration object properties is one way of implementing persistent changes to Waveset behavior.

You can use the Oracle Waveset Integrated Development Environment (Identity Manager IDE) to view and edit Waveset objects for your deployment. Instructions for installing and configuring the Identity Manager IDE are now provided on https://identitymanageride.dev.java.net.

This section describes how to view and edit the following configuration objects:

- "IDM Schema Configuration Object" on page 234
- "UserUIConfig Object" on page 237
- "RepositoryConfiguration Object" on page 237
- "WorkItemTypes Configuration Object" on page 238
- "SystemConfiguration Object" on page 239
- "Role Configuration Object" on page 242
- "End User Tasks Object" on page 245

Note –

- An object's `authType` determines who can view or edit the configuration object.

  To assign an authorization type to an object, you set a new field defined in the `PersistentObject` class. From the Java API, you can access the authorization type using these methods:

  ```
  public void setAuthType(String name);
  public String getAuthType();
  ```

  In the XML for an object, you can set the `authType` attribute in the `root` element. For example:

  ```
  <TaskDefinition name='Request More Space' authType='EndUserTask'
      executor='com.waveset.workflow.WorkflowExecutor' ...>
      ...
  </TaskDefinition>
  ```

- If you change the inline or queryable attributes for `Type.USER`, you must refresh all `User` objects.

  For more information, see "Refreshing User Objects" on page 245.

IDM Schema Configuration Object

You configure User and Role extended, queryable, and summary attributes in the `IDM Schema Configuration` configuration object.

Note – The schema customizations provided in the `IDM ObjectClass Configuration` object are loaded at server startup. Whenever you modify the schema, you must restart the server to load the changes.

Waveset records any problems loading the schema in the system log messages. Use one of the following methods to view these messages:

- Run the `lh syslog` command
- Run the 'Recent System Messages' report from the IDM Administrator Interface (Reports tab)

A sample of the schema can be found in the `schema.xml` file in the sample directory.

Edit the `IDM Schema Configuration` configuration object to add extended attributes to multiple object types during deployment. Specifically, you can

- Configure extended, queryable and summary attributes for Users, Roles, Business Roles, IT Roles, Application Roles, Asset Roles, and any custom roles

- Mark extended and built-in attributes as queryable or summary

---

Note – The IDM Schema Configuration object is protected with the IDMSchemaConfig authType.

Administrators needing to view or edit the Waveset schema for Users or Roles must have the IDMSchemaConfig AdminGroup (capability) assigned. The Configurator user has this AdminGroup assigned by default.

Adding an Extended Attribute to an Object

To add an extended attribute, you must define the attribute with an IDMAttributeConfiguration (unless the attribute is a built-in attribute).

IDMAttributeConfigurations require a name and syntax. The valid syntax options are BOOLEAN, DATE, INT, or STRING. Optionally, an IDMAttributeConfiguration can specify whether the attribute is multi-valued, and can provide a display name (currently not used), and a description.

To add an extended attribute, or mark an attribute (either extended or built-in) as queryable or summary, specify an IDMObjectClassAttributeConfiguration in the appropriate IDMObjectClassConfiguration, such as User. You must specify a name that matches an existing (built-in or configured in the same configuration object) IDMAttributeConfiguration. You can also mark the IDMObjectClassAttributeConfiguration as queryable or summary.

In the following example, firstname, lastname, and fullname are extended attributes. The firstname and lastname User attributes are queryable and summary, but fullname is not.

EXAMPLE 12–1 Extended Attributes Example

```
<?xml version='1.0' encoding='UTF-8'?> <!DOCTYPE Waveset PUBLIC 'waveset.dtd' 'waveset.dtd'>
<Waveset>
<Configuration name="IDM Schema Configuration" id='#ID#Configuration:IDM_Schema_Configuration'
authType='IDMSchemaConfig'>
<IDMSchemaConfiguration>
<IDMAttributeConfigurations>
...
<IDMAttributeConfiguration name='firstname' description='User's first name' syntax='STRING'/>
<IDMAttributeConfiguration name='lastname' description='User's last name' syntax='STRING'/>
<IDMAttributeConfiguration name='fullname' description='User's full name' syntax='STRING'/>
...
</IDMAttributeConfigurations>
<IDMObjectClassConfigurations>
...
<IDMObjectClassConfiguration name='User' extends='Principal'>
...
<IDMObjectClassAttributeConfiguration name='firstname' queryable='true' summary='true'/>
<IDMObjectClassAttributeConfiguration name='lastname' queryable='true' summary='true'/>
<IDMObjectClassAttributeConfiguration name='fullname'/>
```

EXAMPLE 12–1 Extended Attributes Example *(Continued)*

```
...
</IDMObjectClassConfiguration>
</IDMObjectClassConfigurations>
</IDMSchemaConfiguration>
</Configuration>
</Waveset>
```

Note – *To prevent potential conflicts with new core attributes in future releases of Oracle Waveset,* prefix extended attributes with a deployment-specific prefix.

For example, to add an extended attribute to `User` to record the `employeeNumber`, prefer a prefix associated with the company, such as `acme_employeeNumber`. If a future release of Waveset incorporates a built-in user attribute named `employeeNumber`, the two attributes will remain distinct. Otherwise the built-in attribute takes precedence.

Extending the Role Object Class

You can extend Role using an `IDMObjectClassConfiguration`. The following built-in Role extensions all extend the Role object class:

- `BusinessRole`
- `ITRole`
- `AssetRole`
- `ApplicationRole`

To add an extended attribute to a particular role extension, such as `AssetRole`, add the `IDMObjectClassAttributeConfiguration` to the `AssetRole` `IDMObjectClassConfiguration`. To add an extended attribute to all kinds of roles, add the `IDMObjectClassAttributeConfiguration` to the `Role` `IDMObjectClassConfiguration`, and it will be inherited by all extensions of `Role`.

You can define custom extensions of `Role` or any extension of `Role`. For example, to add a custom extension of `AssetRole`, define a new `IDMObjectClassConfiguration` (in the `IDM Schema Configuration`) for the new role, and use the `extends` field to specify the parent role, as shown in the following example:

```
<IDMObjectClassConfiguration name='MyAssetRole'
                             extends='AssetRole'
                             description='My Asset Role Description'/>
```

When you add a new `Role` `objectclass`, you must add a new `Role` type to the `Role Configuration` object. In addition, the new `Role` type's name must match the name of the new `Role` `objectclass`. For more information, see "Role Configuration Object" on page 242.

UserUIConfig Object

Note – You now configure extended, queryable, and summary attributes for Users (`WSUser`) in the schema configuration instead of in the `UserUIConfig` object. For more information, see "IDM Schema Configuration Object" on page 234

The `SummaryAttrRoleCountLimit` attribute controls the number of roles that appear in the summary attribute string for a user. To control this number, specify a value here. If you do not specify a value in this object, Waveset will list at most three roles.

RepositoryConfiguration Object

The `RepositoryConfiguration` object contains settings that control the behavior of the Waveset Repository. Each XML attribute of the top-level `<RepositoryConfiguration>` element configures some aspect of overall Repository behavior.

For example, the following line specifies that repository locks expire in five minutes by default.

```
<RepositoryConfiguration ... lockTimeoutMillis='300000' ... >
```

Caution – *Do not* modify any `RepositoryConfiguration` setting unless you understand its effects.

The `RepositoryConfiguration` object also contains some settings that are specific to `User` objects. For example, the `TypeDataStore` element for `User` objects specifies the *inline attributes* for `User` objects.

Inline attributes are single-valued attributes that the Repository stores directly in the main object table for each type, in this case, in columns `attr1` through `attr5` of the `USEROBJ` table. Most attribute values are stored in the `USERATTR` table (which requires a separate join for each attribute). Inlining an attribute improves the performance of queries that use the attribute.

The sample `RepositoryConfiguration` object specifies default inline attributes for `User` objects, as follows:

```
<TypeDataStore typeName='User' ... attr1='MemberObjectGroups' \
  attr2='lastname' attr3='firstname' attr4='' attr5='' />
```

Do not change the value of `attr1`, which is set to `attr1='MemberObjectGroups'`. You can, however, specify the name of any attribute that is queryable and single-valued as the value of any of the remaining inline columns (`attr2` through `attr5`).

Note –

- If you change the inline attributes for `Type.USER`, you must refresh all `User` objects.

  For more information, see "Refreshing User Objects" on page 245.
- Changes to the `RepositoryConfiguration` object do not take effect until you restart each Waveset server. Restarting an Waveset server restarts the Repository on that server, which causes the Repository to re-read the `RepositoryConfiguration` object.

To view or edit the `RepositoryConfiguration` object, you must have `Debug` and `Security Administrator` capabilities.

For more information, see the "Upgrade Issues" section of the Release Notes, and the *Oracle Waveset 8.1.1 System Administrator's Guide*.

WorkItemTypes Configuration Object

This configuration object is defined in `sample/workItemTypes.xml`, which is imported by `init.xml` and `update.xml`. This object enumerates the supported work item type names, extensions, and display names.

The `extends` attribute allows for a hierarchy of work item types (`workItem` Types). When Waveset creates a work item, it delegates the work item to the specified users if its `workItem` type is:

- The type delegated
- One of the subordinate `workItem` types of the type being delegated

TABLE 12–1 `workItem` Types

Type	extends	Display Name
workItem	none	All Work Items
approval	workitem	Approval
organizationApproval	approval	Organization Approval
resourceApproval	approval	Resource Approval
roleApproval	approval	Role Approval
roleChangeApproval	approval	Role Change Approval
applicationRoleApproval	roleApproval	Application Approval
applicationRoleChangeApproval	roleChangeApproval	Application Change Approval

TABLE 12–1 `workItem` Types *(Continued)*

Type	extends	Display Name
assetRoleApproval	roleApproval	Asset Approval
assetRoleChangeApproval	roleChangeApproval	Asset Change Approval
businessRoleApproval	roleApproval	Business Role Approval
businessRoleChangeApproval	roleChangeApproval	Business Role Change Approval
itRoleApproval	roleApproval	IT Role Approval
itRoleChangeApproval	roleChangeApproval	IT Role Change Approval
attestation	workItem	Access Review Attestation
accessReviewRemediation	workItem	Access
review	workItem	Remediation

SystemConfiguration Object

The `SystemConfiguration` object provides a central control point for many system behaviors and provides a means of storing persistent customizations to system behavior. Given its importance, and the frequency with deployers customize it, the full range of possible customizations are not documented here. Some common customizations are documented here:

- "Controlling the Display of the Password Confirmation Popup" on page 239
- "Configuring Delegate History List Length" on page 240
- "Enabling Attribute Value Customization" on page 240
- "Form and Workflow Save Behavior Customization" on page 240
- "Login-Related Customizations" on page 240
- "PasswordSync-Related Customizations" on page 241
- "Registering Scheduler Startup (for Clustered Environments)" on page 241
- "Source Adapter Task Customization" on page 242

Controlling the Display of the Password Confirmation Popup

The `forgotPasswordChangeResults` attribute in the `System Configuration` object controls whether Waveset displays a confirmation page after a user or administrator has initiated a password change by clicking the Forgot My Password button during log in.

- The default value of `forgotPasswordChangeResults.User` is `true`.
- The default value of `forgotPasswordChangeResults.Admin` is `false`.

Configuring Delegate History List Length

The `delegation.historyLength` attribute controls the size of the list of both current and completed delegations displayed by the End User View workItem Delegation form. This attribute specifies the maximum number of delegations that can appear in the delegation table. Note that the table will show all current delegations, no matter which value you set here.

The `SystemConfiguration` object contains the `security.delegation.historyLength` attribute, which controls the number of previous delegations that are recorded.

Enabling Attribute Value Customization

The `process.handleNativeChangeToAccountAttributes` attribute controls the auditing of attribute values. When set to true, attribute value enabling is enabled for both the reconciliation process and for the provisioner. By default, this property is not enabled.

Form and Workflow Save Behavior Customization

The `security.saveNoValidateAllowedFormsAndWorkflows` attribute lists the IDs of forms and workflows that will be processed as a SaveNoValidate action. All other forms and workflows will be processed as a Save. If this list is not present, the behavior remains the same for all forms and workflows (all forms and workflows will be processed as SaveNoValidate.

Login-Related Customizations

You can customize login behavior by directly editing system configuration object attributes.

Enabling autocomplete for Login Pages

By default, Waveset prevents browsers from offering to store the user's credentials. You can enable the `autocomplete` feature for the login pages by changing the `ui.web.disableAutocomplete` system configuration object to `true`. The login pages include `login.jsp`, `continueLogin.jsp`, `user/login.jsp`, and `user/continueLogin.jsp`.

Waveset login forms other than the preceding ones are generated from XPRESS, and you must edit these forms to use the new display property. These forms, which reside in the sample directory, include this property commented out by default.

- Anonymous User Login
- Question Login Form
- End User Anonymous Enrollment Validation Form
- End User Anonymous Enrollment Completion Form
- Lookup Userid

Displaying an Error Message During an Attempt to Provision a Disabled User

The `ProvisioningDisabledUserShouldThrow` attribute controls whether Waveset will produce an error message when preventing an attempt to provision a disabled user. When set to true, Waveset will prevent any attempt to provision a disabled user to a resource and will produce an error. When this attribute is not set to true, then Waveset will still prevent the provisioning, but will not produce an error.

Launching the Password Login Workflow upon Login

The `runPasswordLoginOnSuccess` attribute controls whether Waveset will run the Password Login workflow when a user successfully logs in. When set to true, Waveset will run this workflow after successful login. By default, the value of this attribute is false.

PasswordSync-Related Customizations

You can customize PasswordSync behavior by directly editing the following system configuration object attributes:

- `PasswordSyncResourceExcludeList` – This attribute controls whether lists of resource names should always be excluded from synchronization.
- `PasswordSyncThreshold` – If PasswordSync is enabled for a resource for which Waveset can also initiate password changes, you can use this setting to prevent a loop-back password change. When you initiate a password change from Waveset, it will set the password on the resource, and the PasswordSync library will notify Waveset of the change. Waveset will then compare the lastPasswordDate on the user object to the current time. If this difference is less than the PasswordSyncThreshold, Waveset will ignore the password change.

Registering Scheduler Startup (for Clustered Environments)

The `scheduler.hosts` attribute registers startup behavior for the scheduler for each Waveset application instance.

The value of `scheduler.hosts` is a map that contains an entry for each host that you want to control. The key is the `hostname` for the Waveset application instance.

Note – To see the `hostname value`, go to the `debug/GetStatus.jsp` page in your Waveset installation.

The following values are valid:

- `enabled` (default)
- `disabled`
- `manual` (suspended)

The default value is used if no value or an invalid value is specified.

Note – The `task.scheduler.enabled` and `task.scheduler.suspended` properties in the `Waveset.properties` file override the value set in the System Configuration object.

Following is an example of the `scheduler` attribute from `Configuration:System` Configuration:

```
<Attribute name='scheduler'>
   <Object>
      <Attribute name='hosts'>
         <Map>
            <MapEntry key='goliad' value='enabled'/>
            <MapEntry key='sanjacinto' value='manual'/>
            <MapEntry key='washington' value='disabled'/>
         </Map>
      </Attribute>
   </Object>
</Attribute>
```

Source Adapter Task Customization

You can edit the following two attributes to customize the behavior of the source adapter task:

- `sources.subject` – Specifies the login name of the administrator designated as the owner of the source adapter task.
- `sources.hosts` – Specifies the server on which the source adapter task runs.

Role Configuration Object

The `Role Configuration` object defines the supported Role Types, Actions, and List Columns. The following sections describe the supported elements of a Role Type definition:

- "Types Attribute" on page 242
- "Actions Attribute" on page 244
- "List Columns Attribute" on page 245
- "Other Options" on page 245

Types Attribute

Role type attributes are configured in the `types` section of the `Role Configuration` object. For each type of role in the list, for example business or IT roles, you must specify the following attributes:

- "displayName Attribute" on page 243
- "authType Attribute" on page 243
- "workItemTypes Attribute" on page 243
- "features Attribute" on page 243

displayName Attribute

Specifies the type's display name whose value is a message catalog key.

authType Attribute

Specifies the authorization type associated with the role type. An authorization type enables fine-grain authorization for who is allowed to view and manage this role type. If you have not yet defined an `authType`, add one to the `AuthorizationTypes` configuration object. You must reference that `authType` within an `AdminGroup` (capability) as a type within a `Permission` that grants access to roles of this `authType`.

Note – All roles have an authorization type. If you load a role without an authorization type, the authorization type defaults to `ITRole`.

---

workItemTypes Attribute

The type of work items that can be created for role assignment approval and role change approval. If you have not yet defined the specified `workItem` types, add them to the `WorkItemTypes` configuration object.

features Attribute

The `features` attribute includes the following features:

- `changeApproval`. If specified, indicates that Owners specified in the Role must approve any changes to a Role of this type. If no Owners are specified, then no approvals occur.
- `changeNotification`. If specified, indicates that any changes to a Role of this type will send email notifications to the owners of the specified Role.
- `containedTypes`. Required feature whose value is the list of Role types that can be contained in this type, where the allowed values are:
  - `BusinessRole`
  - `ITRole`
  - `ApplicationRole`
  - `AssetRole`
  - Custom role types
- `assignResources`. If specified, indicates that resources and resource groups can be assigned to roles of this type. If not specified, defaults to no Resources can be assigned to Roles of this type.
- `userAssignment`. If specified, indicates whether Roles of this type can be directly assigned to Users. If this Role type can be assigned directly to Users, this feature also specifies whether the Users can be assigned manually and automatically. If not specified, defaults to user assignment not allowed.

Note – Automatic assignment is not supported in this release, but will be in a future release.

- `manual`. If specified (for example `true` or `false`), indicates whether you can manually assign Roles of this type to Users.
  - `activateDate`. If specified (for example `true` or `false`), indicates whether you can specify a future activation (start) date for Roles of this type when assigned to a User. Note that this feature is valid only if `userAssignment.manual` is `true`.
  - `deactivateDate`. If specified (for example `true` or `false`), indicates whether you can specify a future deactivation (end) date for Roles of this type when assigned to a User. Note that this feature is valid only if `userAssignment.manual` is `true`.

Note – You can set both `activateDate` and `deactivateDate` to `true`, even if `userAssignment.manual` is not. If you set both attributes to `true` for a roleType, and if the role is contained by another role optionally, then you can specify activate and deactivate dates when assigning the optional role to a user.

- `roleExclusions`. If specified, indicates that Roles of this type allow the Role editor to specify a list of Roles that cannot be assigned to a user if this Role is assigned; an exclusion list.

Actions Attribute

The Actions attribute defines a set of actions that a Role administrator can take on one or more Roles in the list Roles table and when adding role exclusions to contained roles to an existing role.

Three sets of actions are specified in role configuration:

- `actions`. Actions displayed in the main role list and on the Find Role Results pages.
- `addContainedRoleActions`. Actions displayed as an administrator is adding contained roles to a role.
- `addRoleExclusionsActions`. Actions displayed as an administrator is adding a role exclusion to a role.

Each action is defined with the following attributes:

- `action`. Specifies the command.
- `label`. Specifies the display name message key.
- `requiredPermissions`. Permissions that control whether the action is displayed, depending on the administrator's permissions.
  - `Type`. Type of object to which an administrator must have the given rights.
  - `Rights`. List of rights that an administrator must have for the given object type

- `selectionRequired`. Indicates that a role must be selected for this action.
- `type`. Specifies the role action type, which can be `create`, `update`, `delete`, or `task`.
- `view`. Copies the contents of this attribute onto the role view during the execution of the action for `create`, `update`, and `delete` role action types.
- `task`. Specifies the task to launch for task action types.
- `skipTaskLaunchForm`. If set to `true`, skips the task launch form. Otherwise the task launch form (if present) is displayed. Applies to `task` action types.

List Columns Attribute

The List Columns attribute defines the set of attribute names and labels to display as column headings when viewing lists of Roles (for example, List roles and find role results).

You can specify unique sets of attributes to display as list column headings. The attributes for each defined column are

- `name`. Name of the role attribute to display
- `displayName`. Display name to appear in the column header
- `rule`. Optional rule that might format the attribute value. The `rule` is invoked for each row in the list, and the value returned by the rule is what displays in each table cell.

Other Options

You can also set the following options in the Role Configuration object:

- `roleListMaxRows`. The maximum number or roles to list
- `roleListPageSize`. The number of roles to display on a single page

End User Tasks Object

The `End User Tasks` object defines the tasks that you can run from the Waveset user interface. You can assign the `EndUserTask` authorization type to any `TaskDefinition` object, and you can assign the `EndUserRule` authorization type to any `Rule` objects that must be exposed.

Refreshing User Objects

Certain types of changes require an administrator to refresh all `User` objects. For example, you must refresh all `User` objects when you change the inline attributes for `Type.USER` in `RepositoryConfiguration`. Whenever you mark an attribute as queryable or summary in the `IDMSchemaConfiguration` object, you must refresh all `User` objects for the change to affect older, unmodified objects. The same logic applies when a new version of Waveset adds a new attribute, or when a new version of Waveset changes the values of an existing attribute. The upgrade process or an administrator must refresh all `User` objects for the change to affect older, unmodified objects.

There are three ways to re-serialize existing users:

- Modify an individual User object during normal operations.

  For example, opening a user account through the user interface and saving it with or without modifications.

  Disadvantage: This method is time-consuming, and the administrator must be meticulous to ensure all existing users are re-serialized.
- Use the `lh refreshType` utility to re-serialize all users. The `refreshType` utility's output is a refreshed list of users.

  ```
  lh console
  refreshType User
  ```

  Disadvantage: Because the `refreshType` utility runs in the foreground and not the background, this process can be time-consuming. If you have a lot of users, re-serializing them all takes a long time.
- Use the Deferred Task Scanner.

Note – Before running the Deferred Task Scanner process, you must edit the `System Configuration` object using the Identity Manager IDE or some other method.

Search for '`refreshOfType`' and remove the attributes for '`2005Q4M3refreshOfTypeUserIsComplete`' and '`2005Q4M3refreshOfTypeUserUpperBound`'.

After editing the `System Configuration` object, you must import that object to repository for your changes to be present.

Disadvantage: This method causes the next Deferred Task Scanner run to take a long time because it examines and rewrites almost every User object. However, subsequent Deferred Task Scanner runs should execute at normal speed and duration.

CHAPTER 13

Enabling Internationalization

This chapter describes how to configure Waveset to use multiple languages or to display a language other than English.

Architectural Overview

TABLE 13–1 Components of Waveset Internationalization

File	Description
`WPMessages.properties`	Default message file located in `$WSHOME/idm/web/WEB-INF/classes/com/waveset/msgcat`. Shipped as part of the `idmcommon.jar` file. Displays message text in English and loads by default unless you've customized your Waveset installation to behave otherwise.
`Waveset.properties`	Located in `$WSHOME/config`. To enable support for multiple languages, you must set `Internationalization.enabled` to true. (*Default is true.*)
System Configuration Object	Specify a custom message catalog
Additional message file for each supported language	Additional supported languages each require their own message file. `WPMessages_`*xx*`_`*XX*`.properties`, where *xx* represents the language and *XX* represents the country. For example, `WPMessages_en_US.properties` contains messages in American English. Each international catalog has its own `.jar`.

Note –

- If you loaded a new catalog in `/config` that uses the same name as the default catalog, the new catalog takes precedence over the default.
- If you have more than one message file, you can specify the catalog from which a message key is derived by specifying `catalogname:keyname`.

The following catalog entries control how the product name is displayed:

```
PRODUCT_NAME=Waveset
LIGHTHOUSE_DISPLAY_NAME=[PRODUCT_NAME]
LIGHTHOUSE_TYPE_DISPLAY_NAME=[PRODUCT_NAME]
LIGHTHOUSE_DEFAULT_POLICY=Default [PRODUCT_NAME] Account Policy
```

Typical Entry

Messages are contained in key/text pairs and contain three parts:

- A text string, or key, that is an identifier used by the code to retrieve data. Do not translate this required component. This component is used in the product configuration, and acts as a placeholder for the translation.
- An equals (“=”) sign separating the key and text. This entry is required.
- A string containing data that is displayed when running the application. This entry is the translation, used in place of the key whenever the page is rendered in the browser.

Each line in the resource array contains two strings. Translate the second quoted string on each line.

Certain strings to be translated contain special codes for data that is inserted into the string when it is displayed. For example, if you have the following string to translate:

```
UI_USER_CONNECT={0}, connected at 100 mbs
```

the rendered version could appear as `jfaux, connected at 100 mb`

Translations typically appear inside a browser, so it is appropriate to add HTML tags to format the string, as shown below:

```
_FM_ACCOUNT_ID_HELP=<b>Account ID</b><br>Enter a name for this user. This field is required.
```

Enabling Support for Multiple Languages

Use the instructions described in this section to enable multiple language support for Waveset.

Step One: Download and Install Localized Files

Perform the following tasks”

- “Before You Install” on page 249
- “Download Message Catalog Files” on page 249
- “Install Localized Files” on page 250

Before You Install

Perform the following tasks before you install localized files:

1. Install Waveset. See Installation Guide for detailed installation procedures.
2. Make sure the following locales on the application server have been set to UTF-8
   - Application server instance
   - Database
   - Java Virtual Machine (JVM)

     Refer to the documentation for these products for information about setting the locale.

Download Message Catalog Files

The Waveset software download website provides the following localized message catalogs. Download the appropriate message catalog jar file and place that file in the `WEB-INF/lib` directory.

TABLE 13–2 Message Catalog Files

File Name (.zip)	Language	Locale
IDM__8_1_l10n_de	German	de_DE
IDM__8_1_l10n_es	Spanish	es_ES
IDM__8_1_l10n_fr	French (France and Canada)	fr_FR
IDM__8_1_l10n_it	Italian	it_IT
IDM__8_1_l10n_ja	Japanese	ja_JP
IDM__8_1_l10n_ko	Korean	ko_KR
IDM__8_1_l10n_pt	Brazilian Portuguese	pt_BR

TABLE 13–2 Message Catalog Files *(Continued)*

File Name (.zip)	Language	Locale
IDM__8_1_l10n_zh	Simplified Chinese	zh_CN
IDM__8_1_l10n_zh_TW	Traditional Chinese	zh_TW

Download the ZIP file to a temporary location. By default, the contents of the ZIP file are extracted to the `FileName\IDM_8_0_l10n` directory, where `FileName` matches the name of the downloaded file, minus the ZIP extension.

Zip File Contents

Every extracted ZIP file contains the following:

- A JAR file containing localized message catalogs, help files, and other essential files. The JAR file is named `IDM_8_1_l10n_Locale.jar`.
- Waveset Localization README

Additional translated publications might also be available.

Install Localized Files

Copy the JAR file from the temporary location to the `IdentityManagerInstallation/WEB-INF/lib` directory.

Step Two: Edit the `Waveset.properties` File

To edit the `Waveset.properties` file,

▼ To Edit the File

1 **Open the `InstallDir/config/Waveset.properties` file with your editor of choice.**

2 **Change the `Internationalization.enabled` property to `true`.**

3 **Save your changes and close the file.**

4 **Either restart Waveset or click Reload Properties on the Debug pages available at the following location:**

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

Maintaining ASCII Account IDs and Email Addresses During Anonymous Enrollment Processing

By default, Waveset's anonymous enrollment processing generates values for `accountId` and `emailAddress` by using user-supplied first (`firstName`) and last names (`lastName`) as well as `employeeId`. Because anonymous enrollment processing can result in the inclusion of non-ASCII characters in email addresses and account IDs, international users must modify EndUserRuleLibrary rules so that Waveset maintains ASCII account IDs and email addresses during anonymous enrollment processing.

To maintain account ID and email address values in ASCII during anonymous enrollment processing, follow these two steps:

▼ To Maintain Account ID and Email Address Values

1 **Edit the following rules in the EndUserRuleLibrary as indicated in the following table:**

Edit this rule	To make this change...
`getAccountId`	To use `employeeId` only (and remove `firstName` and `lastName`)
`getEmailAddress`	To use `employeeId` only (remove `firstName`, `lastName`, and ".")
`verifyFirstname`	To change length check from 2 to 1 to allow for single character Asian first names

2 **Edit `EndUserAnonEnrollmentCompletionForm` to remove the `firstName` and `lastName` arguments from calls to the `getAccountId` and `getEmailAddress` rules.**

Index

Numbers and Symbols

@todo, 178

A

account attributes
    defining, 157, 162-163
    description, 162-163
    mapping resource attributes, 178, 179
    standard Waveset, 162-163
    using Correlation rule, 153
    using process rule, 154
    using Resolve Process rule, 154
account DN, 164
account ID policies, 68-69
account index, 35
    bulk actions and, 67
    linking accounts with, 74-75
    load from file and, 66
    load from resource and, 66
    reconciliation and, 67
account name syntax, 164
account reconciliation, 38-39
AccountAttribute, 157
AccountAttributesTypes, 179
accountID, 153, 168
accountId, 157, 163
accounts
    disabling, 187
    enabling, 184, 187
Active Directory, 62, 76, 77
Active Directory *(Continued)*
    Users and Computers MMC, 65
Active Sync
    IAPIProcess, 51, 154
    IAPIUser, 50, 154
    interface, 150
    loading account data, 64
    overview, 156
    resource attributes, 160
Active Sync-enabled adapters
    event-driven, 172
    identifying Waveset user, 152
    initializing, 189
    methods, writing, 189-191
    overview, 149
    polling, 172, 190-191
    storing and retrieving attributes, 191
    updating the Waveset repository, 191
ActiveSyncUtil class, 189
adapters
    account disabling example, 56-59
    Active Sync-enabled.
        See Active Sync-enabled adapters
    build environment, 174-176
    configuring, 68
    creating custom, 147, 176-177
    debugging, 200-219
    defining features, 183-187
    defining resource forms, 198-199
    experience requirements, 148
    form processing, 53-59
    important notes, 148

adapters *(Continued)*
initializing, 189
installing custom, 199-200
maintaining custom, 219
methods.
See methods, adapter
overview, 149-156
recommended reading, 148
registering, 151
sample files for creating, 171, 173-174
scheduling, 190-191
setting options and attributes, 179-180
standard, 149
testing custom, 200-219
writing methods, 180-191
AdminGroups, 30
AIXResourceAdapter.java file, 174
APIs, registering adapters, 151
APIs, Waveset, 219
Attribute Condition Operators, 23-24
attribute conditions, 22
AttributeDefinitionRef, 179
attributes
account.
See account attributes
custom, 179
description, 17
extended, 19
extended schema, 171
Identity System, 21
inline, 18
managing, 197
mapping syntax, 179
operational, 20
other, standard, 22
queryable, 18
resource.
See resource attributes
Resource User, 21
secret, 25
summary, 18
types, 17
user, 155
user view, 65, 70

attributes *(Continued)*
view, 21
Waveset, 155
authenticate() method, 188
authentication
pass-through.
See pass-through authentication
AuthnProperty, 168
authorization types
creating, 31
description, 27
using, 28
`AuthType` elements, 29

B
build environment for adapters, 174-176
built-in attributes, 20
exposing, 20
marking as queryable or summary, 235
precedence, 19
bulk action, creating, 66-67, 85
bulk operations, 41

C
calling
methods, 181, 190
capabilities
adding to roles, 119
assigning, 31, 32
Debug, 238
defining, 30-31, 155
discovery and reconciliation, 40
EndUser, 30
granting, 119
Security Administrator, 238
classes
ActiveSyncUtil, 189
editing public, 156
IAPI, 191
Java, 181
object, 193

classes *(Continued)*
recompiling, 219
resource adapter, 155
ResourceAdapterBase, 149, 183
comma-separated values file., *See* CSV file
`Configuration`, AuthorizationTypes object, 29
configuration, login, 157
configuring an adapter, 68
confirmation rule, 152
confirmation rules
custom, 73-74
linking accounts with, 71
load from file and, 65
load from resource and, 66
connecting with resource, 181
connection information, 155
connection settings, checking, 182-183
correlation keys, custom, 72-73
correlation rule, 153
correlation rules
custom, 73-74
linking accounts with, 71
load from file and, 65
load from resource and, 66
create bulk action, 66-67, 85
create requests, 194
Create Unmatched Accounts, 153
CSV file, 65, 66, 85
custom adapter
installing, 199-200
maintaining, 219
recompiling, 219
testing, 200-219
custom attributes, 179
custom correlaton keys, 72-73
custom rules, 73-74
customizations file, 129
customizing
Data Exporter, 97-101
headers and footers, 129-130
logo, 135-136
Waveset pages, 130
customStyle.css, 128, 129

D
Data Exporter
architecture, 89-90
connection pooling, 91
customizing, 97-101
data types, 88-89
database requirements, 91-93
export schema, 98-99
export server, 93
factory classes, 100-101
Hibernate support, 91
logging, 102
ObjectClass schema, 97-98
overview, 87-88
planning for, 91-93
space requirements, 93
task, 93
tracing, 102
troubleshooting, 101-102
Warehouse Interface Code, 99
data loading
account, creating, 69
preparing for, 68-71
processes, 64-67
types, 40-41
data types, 88-89
data types, supported, 88
database accounts adapter files, 174
database tables adapter files, 174
DB2 DDL, 94
Debug pages, Waveset, 219
debugging custom adapters, 200-219
default style settings, 129
default text, 128
`delegation.historyLength` attribute, 240
delete rule, 153
deletion requests, 153
distinguished name, setting, 180
documentation, related, 148

E
email account attributes, 163

enabling
  accounts, 184
  Pass-Through Authentication, 188
EndUser Capability, 30-31
environment, assessing, 61
example labeling exercises, 135-141
examples
  login configuration, 169
  object resource attribute declarations, 169
  object type definitions, 195
ExampleTableResourceAdapter.java file, 174
experience requirements, for developing custom adapters, 148
export attribute parameters, 98-99
export schema, 98-99
export task, 93
extended attributes, 234
  adding to ObjectClass schema, 100
  description, 19
  marking as queryable or summary, 235
  where supported, 20
extended schema attributes, 171, 179, 180
Extended User Attributes Configuration object, 72
extending Role objectClass, 236

F
features
  account, 183-184
  general, 183
  getFeatures() method, 183
  group, 184-185
  organizational unit, 185
file-based accounts adapter files, 174
find requests, 218
font characteristics, changing, 134-135
footer
  chanigng bar colors, 136
  customizing, 129-130
forensic queries, 90
`forgotPasswordChangeResults attribute` attribute, 239
form objects, designating, 157
forms
  assigning, 198-199
  referencing, 218

G
getFeatures() methods, 183, 188

H
header
  changing bar colors, 136
  customizing, 129-130
header information, adapter source code, 156
Hibernate support, 91

I
IAPI classes, 191
IAPI objects, 151, 191
IAPIFactory.getIAPI methods, 151, 191
IAPIProcess, 51, 154, 191
IAPIUser, 50, 154, 191
Identity Application Programming Interface (*IAPI*), 151
Identity System attributes, description, 21
identity template, 157, 164-165, 168
IDM Schema Configuration configuration object, 234
important notes, for developing custom adapters, 148
init() method, 189
initializing an adapter, 189
inline attributes, description, 18
installing
  custom adapters, 199-200
  REF Kit, 175

J
Java
  classes, 181
  defining resource attributes, 158

Java *(Continued)*
header information, 156
resource adapters, 149, 174
Java classes, recompiling, 219
JAVA_HOME, 175, 176
JavaDocs, 173
JSP files, 129

L
labeling exercises, sample, 135-141
language support, enabling, 249-250
language support,maintaining ASCII account IDs and email addresses during anonymous user enrollment, 251
LDAP, 62, 80, 82-84
LDAP-based resource objects, 193
linking accounts
manually, 74-75
overview, 71
using account index, 74-75
using self-discovery, 75
list method, 186
load from file, 41
Load from File, 64
load from resource, 41
Load from Resource, 64, 66
load operations, 41
loading data, example scenarios, 76-86
login configuration, 157, 166, 168-169, 169
LoginConfigEntry, 157, 168, 169, 188
logo, customizing, 135-136

M
managing
attributes, 197
groups and organizations, 167
resources, 149, 151, 155, 158
map, schema., *See* schema map
mapping, resource attributes, 179
MBeans, 93
message catalog files, 249-250
messages, internationalizing, 248
methods
calling, 181, 190
getFeatures(), 183
IAPIFactory.getIAPI, 151, 191
startConnection, 181
stopConnection, 181
methods, adapter
Active Sync-specific, 189-191
checking connections and operations, 182-183
connecting with resource, 181
creating an account on a resource, 185
creating the prototype resource, 181
defining features, 183-187
deleting accounts on a resource, 185-186
enabling and disabling accounts, 187
enabling pass-through authentication, 188
getting user information, 186
initializing and scheduling the adapter, 189
list methods, 186
overview, 166-167
polling the resource, 190-191
standard resource adapter-specific, 180-189
storing and retrieving adapter attributes, 191
updating accounts on the resource, 186
updating the Waveset repository, 191
writing, overview, 180-191
MMC, 65
MySQL DDL, 94
MySQLResourceAdapter.java file, 174

N
native disable utilities, 187

O
object attributes, 197-198
object classes, 193-194
object features, 196
object types, 195
ObjectAttributes, 197
ObjectClass schema, 97-98

ObjectClass schema *(Continued)*
adding custom attributes, 100
description, 97
extending for User and Role types, 97
ObjectClasses, 194
ObjectFeatures, 196
objects
`Configuration`
AuthorizationTypes, 29
resource.
See resource objects
ObjectTypes, 195
operational attributes, description, 20
Oracle DDL, 95
other, standard attributes, description, 22

P
page title and subtitle, changing, 131-132
pass-through authentication, 166, 168-169, 188
password attributes, 163
password policies, 68-69
PeopleSoft, 62, 80, 81-82
poll() method, 190
polling a resource, 190-191
polling scenarios, 190
Populate Global, 153
process rule, 154
properties
authentication, 168
Waveset.properties, 174
prototype resource, creating, 181
prototypeXML
description/purpose, 156
resource type, 158
standard resource adapter issues, 167-170

Q
queryable attributes, 234
description, 18
quick links, adding to login page, 131

R
README files, 174
recommended reading, related to adapters, 148
recompiling custom resource adapters, 219
reconcile configuration object, 48-49
reconciliation, 38-39
auditing native changes, 47
confirmation rules, 43-45
correlation rules, 43-45
daemon task, 48
overview, 64, 67
policy settings, 42-47
resource schedules, 48
workflows, 45-47
reconciliation process, 42
reconRules.xml, 73
REF Kit
files/directories, 173
installing, 175
location, 173
sample adapter files, 173
sample files, 173-174
referencing, forms, 218
related documentation, 148
RelationalDataStore, 24
Remedy, 80, 84-85
repository, updating, 191
requests
create, 194
deletion, 153
find, 218
update, 194
requirements, experience, for developing custom adapters, 148
resolve process rule, 154
resource
adapters.
See adapters
attributes
Active Sync-specific, 160
defining, 158
mapping to account attributes, 179
overview, 157, 158
choosing initial to load, 62-64

resource *(Continued)*
  connecting with, 181
  creating account on, 185
  forms, 198-199
  instance, creating, 181
  methods.
    See methods, adapter
  objects, 155
    attributes, 197-198
    classes, 193-194
    features, 196
    LDAP-based, 193
    non-LDAP-based, 194
    testing in Waveset, 218-219
    types, 195
    viewing, 217
  schema map.
    See schema map
  XML definition, 156
resource adapter classes, 155
resource attributes
  defining, 158
  mapping account attributes, 179
  mapping extended schema attributes, 180
  mapping to account attributes, 178
Resource Extension Facility kit., *See* REF Kit
resource objects
  defining capabilities, 155
  description/purpose, 155
resource timeout settings, 48-49
Resource User attributes, description, 21
ResourceAdapterBase class, 149, 183
ResourceAttribute, 157, 158
resources, managing, 149, 151, 155, 158
Role
  defining custom extensions, 236
  extending the objectClass, 236
Role types, ObjectClass schema, 97
Roles
  adding new capabilities, 119
  configuring attributes, 234
roles
  controlling the number of, 237
  description, 88
Roles
  editing schema for, 20, 235
  extended attributes, 20
roles
  viewing, 31
rules
  Correlation, 153
  custom, 73-74
  process, 154

S
SAP, 62
scenarios, polling, 190
`scheduler.hosts` attribute, 241
scheduling an adapter, 190-191
scheduling parameters, 190
schema, editing, 20, 235
schema map, 164, 179
schemas
  export, 98-99
  ObjectClass, 97-98
secret attributes, 25
SecurID, 76, 78
self-discovery, 75
servers, connection settings, 155
skeleton files, adapter
  editing, 178
  login configuration, 169
  overview, 177
Solaris, 76, 78-80
source code for adapters, 156
special considerations, for developing custom
  adapters, 148
SQL Server DDL, 95
standard adapters, 149
  See adapters
startConnection methods, 181
stopConnection methods, 181
style.css, 128
style settings, default, 129
style sheets, modifying, 128
summary attributes, 234
Sun Resource Extension Facility Kit., *See REF Kit.*

supported, data types, 88
SupportedApplications, 168, 169
syntax
  account name, 164
  mapping attributes, 179
  view path, 41
system configuration object, internationalizing, 247
`SystemConfiguration` object, 239

T
Template, 157
testing
  custom adapters, 200-219
  resource object in Waveset, 218-219
text, default, 128
text attributes, 128-129

U
UNIX accounts adapter files, 174
update requests, 194
updating accounts on a resource, 186
user attributes
  defined by resource objects, 155
  retrieving, 186
user forms, 69-71
user identity template.
  See identity template
User Name Matches AccountId, 65, 66
user names, 165
user view, 70
  attributes, 65
Users
  editing schema for, 20, 235
utilities, native disable, 187

V
view attributes, 70
  description, 21
view path syntax, 41
viewing objects, 31

W
Warehouse Interface Code, 99
  generating classes, 100-101
Waveset
  account attributes.
    See account attributes
  attributes, 155
  logo, replacing, 135-136
  password policies, 68-69
  repository, 191
  standard account attributes, 162
waveset.accountId, 70
Waveset Debug pages, 219
waveset.organization, 70
Waveset.properties, 174
Waveset.properties file, 247, 250
Wavesetr, user, identifying, 152
WIC source code, 100
workItem Types, 238-239
WPMessages_en.properties, 129, 132
WPMessages.properties file, 247
WSHOME, 175, 176

X
XML
  resource definition.
    See prototypeXML.
XML files, 65
XMLResourceAdapter.java file, 174