Sun™ Identity Manager 8.0 Technical Deployment Overview

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

Part No: 820-2961-10

Contents

Preface

This *Sun Java™ System Identity Manager Technical Deployment Overview* publication provides an overview of the reference and procedural information you will use to customize Sun Java™ System Identity Manager for your environment.

Who Should Use This Book

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

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

How This Book Is Organized

Identity Manager Technical Deployment Overview 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 Identity Manager deployment.
- Chapter 2, Data Loading and Synchronization — Presents an overview of the reconciliation and other mechanisms for loading account information into Identity Manager. Reconciliation compares the set of users defined in Identity Manager to the set of accounts that are defined on an Identity Manager resource.
- Chapter 3, Dataloading Scenario — Provides tips to consider when preparing to load account information into Identity Manager, including sample scenarios that illustrate some of the issues that you might encounter.
- Chapter 4, Data Exporter — Describes how to plan for and implement the Data Exporter feature.
- Chapter 5, Configuring User Actions — Details how to add custom tasks to the Identity Manager Administrator Interface and configure user actions that you can execute from two areas of the interface.
- Chapter 6, Private Labeling of Identity Manager — Describes how to customize IDM colors, logos, and header and footer content to meet the style standards of your organization.
- Appendix A, Editing Configuration Objects — Provides an overview of configuration objects and a discussion of the `UserUIConfig` object.
- Appendix B, Enabling Internationalization — Provides information on configuring Identity Manager to use multiple languages or display a language other than English.

Conventions Used in This Book

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

- Typographic Conventions
- Symbols
- Shell Prompts

Typographic Conventions

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

Table 1 Typographic Conventions

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

Symbols

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

Table 2 Symbol Conventions

Symbol	Description	Example	Meaning
`[ ]`	Contains optional command options.	`ls [-l]`	The `-l` option is not required.
-	Joins simultaneous multiple keystrokes.	Control-A	Press the Control key while you press the A key.
+	Joins consecutive multiple keystrokes.	Ctrl+A+N	Press the Control key, release it, and then press the subsequent keys.
>	Indicates menu item selection in a graphical user interface.	File > New > Templates	From the File menu, choose New. From the New submenu, choose Templates.

Shell Prompts

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

Table 3 Shell Prompts

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

Related Documentation and Help

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

- *Identity Manager Installation*: Step-by-step instructions and reference information to help you install and configure Identity Manager and associated software.
- *Identity Manager Upgrade*: Step-by-step instructions and reference information to help you upgrade and configure Identity Manager and associated software.
- *Identity Manager Administration*: Procedures, tutorials, and examples that describe how to use Identity Manager to provide secure user access to your enterprise information systems.
- *Identity Manager Deployment Tools*: Reference and procedural information that describes how to use different Identity Manager deployment tool. This information addresses rules and rules libraries, common tasks and processes, dictionary support, and the SOAP-based web service interface provided by the Identity Manager server.
- *Identity Manager Workflows, Forms, and Views*: Reference and procedural information that describes how to use the Identity Manager workflows, forms, and views — including information about the tools you need to customize these objects.

- *Identity Manager Resources Reference*: Reference and procedural information that describes how to load and synchronize account information from a resource into Sun Java™ System Identity Manager.
- *Identity Manager Tuning, Troubleshooting, and Error Messages*: Reference and procedural information that provides guidance for tuning Sun Java™ System Identity Manager, provide instructions for tracing and troubleshooting problems, and describe the error messages and exceptions you might encounter as you work with the product.
- *Identity Manager Service Provider Edition Deployment*: Reference and procedural information that describes how to plan and implement Sun Java™ System Identity Manager Service Provider Edition.
- *Identity Manager Help*: Online guidance and information that offer complete procedural, reference, and terminology information about Identity Manager. You can access help by clicking the Help link from the Identity Manager menu bar. Guidance (field-specific information) is available on key fields.

Accessing Sun Resources Online

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

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

Contacting Sun Technical Support

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

- The online support Web site at http://www.sun.com/service/online/us

- The telephone dispatch number associated with your maintenance contract

Related Third-Party Web Site References

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

Sun Welcomes Your Comments

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

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

For example, the title of this book is Sun Java™ System *Identity Manager Technical Deployment Overview*, and the part number is 820-2961-10.

Chapter 1

Working with Attributes

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

Related Chapters

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

- *Views* chapter of *Identity Manager Workflows, Forms, and Views* provides an extensive discussion of view attributes, registering attributes, and deferred attributes.
- *Identity Manager Resources Reference* provides information about resource attributes.

What are Attributes?

Attributes are name-value pairs that are used to define and manipulate characteristics of Identity Manager objects as well as external resources. Identity Manager 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 Identity Manager deployment can contain a variety of attributes, you typically work with the attribute types described in the following sections:

- Summary Attributes
- Queryable Attributes
- Inline Attributes
- Extended Attributes
- Operational Attributes
- View Attributes
- Resource User Attributes
- Identity System User Attributes
- Other Standard Attributes

Summary Attributes

Every persistent object exposes a set of *summary attributes*. Summary attributes are configured in the Identity Manager 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 Identity Manager Schema Configuration object. For more information, see "`IDM Schema Configuration` Object" on page 143.

NOTE The `SummaryAttrNames` section of `UserUIConfig` is no longer used in Identity Manager.

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 Identity Manager Schema Configuration object. Queryable attributes can be multi-valued.

NOTE The `QueryableAttrNames` section of `UserUIConfig` is no longer used in Identity Manager.

Inline Attributes

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

Identity Manager 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`.

Identity Manager stores the value of an *inline* attribute, however, directly in the object table for that type. Designating an attribute as inline allows Identity Manager 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 Identity Manager.

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 Identity Manager 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 (`string`, `int`, `date`, etc.), 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 Identity Manager 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 *Identity Manager Workflows, Forms, and Views*.

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 Identity Manager 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

Identity Manager 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 Identity Manager 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 Identity Manager 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 *Identity Manager Workflows, Forms, and Views*.

Resource User Attributes

Resource User attributes map Identity Manager 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 *Identity Manager Resources Reference*.

Identity System User Attributes

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

Working with Identity System User attributes is extensively documented in *Identity Manager Resources 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, date created, etc.).

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 Identity Manager 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, Identity Manager 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 Identity Manager's standard authorization scheme.

Using Attribute Conditions

An *attribute condition* is an expression that tests the value(s) 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 operators including:

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.

Table 1-1 Attribute Condition Operators *(Continued)*

Operator	Description
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 ANDed

A set of attribute conditions is implicitly ANDed. 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.

Identity Manager attribute conditions expose operators that are generally useful. Typically, you can express a set of selection criteria using Identity Manager 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 Identity Manager) 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 Identity Manager user because more than one Identity Manager user might have the same acctid but on different resources.
- **Identity Manager 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

Identity Manager 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 Identity Manager 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, Identity Manager 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 Identity Manager deployments. Topics in this chapter include

What are Authorization Types?

Identity Manager 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 vastly different functions that consequently require different authorization.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 Identity Manager administrators with a specific capability.

How Identity Manager Uses Authorization Types

Identity Manager 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'/>**

This second approach is much harder to configure because administrators with rights on the parent types (red, green, blue) will also have access to type 'foo'.

Architectural Features

The primary architectural feature of authorization types is the `Configuration:AuthorizationTypes` object. You can add or remove authorization types by modifying this object.

`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.

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 an authorization type repository type that is the supertype of this type.
displayName	Provides an alternate display name for this type, typically a message catalog key.
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 Subtype Permissions

Identity Manager uses the `extends` attribute to define the supertype of an authorization type. Supertype permissions are inherited by the subtype. For example, if a user has view rights on `TaskDefinition`, they would also have view rights on `UsageReportTask` and all other subtypes of `TaskDefinition`.

Although you can edit the `AuthorizationTypes` object only in XML, you can define permissions that reference authorization types from the Capability page. (You can access this page under the Capabilities subtab of the Security tab.)

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

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.

Code Example 2-1

```
<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.

Identity Manager 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. Identity Manager 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

Identity Manager 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`.

Code Example 2-2 Defining a Resource-Specific Authorization Type

```
<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 Identity Manager

You can also use authorization types to grant fine-grained administrative control of a very specific part of Identity Manager 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 Identity Manager.

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

- **User** — A virtual identity that is managed by Identity Manager. An Identity Manager 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 Identity Manager). 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 Identity Manager system.

Identity Manager 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 Identity Manager 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 Identity Manager 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

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

- Discovery — Provides basic functions that initially load resource accounts into Identity Manager.
- Reconciliation — Periodically loads resource account information into Identity Manager, 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 Identity Manager 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 Identity Manager 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, Identity Manager 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 Identity Manager user from any input account that does not match.

Identity Manager provides the following discovery functions:

- Load From File — Reads accounts listed in a file and loads them into Identity Manager.
- Load From Resource — Extracts accounts from a resource and loads them directly into Identity Manager.
- 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 Identity Manager accounts. See *Identity Manager Administration* 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. Identity Manager 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 Identity Manager and resource accounts, but this discussion will be limited to Identity Manager creating accounts. See *Identity Manager Administration* 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 Identity Manager 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 Identity Manager user.
- **command** — Contains the action taken on the Identity Manager user. For creating Identity Manager 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 *Identity Manager Workflows, Forms, and Views* 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 Identity Manager account.
- `waveset.resources` — A list of one or more resource names to assign to the Identity Manager account.
- `waveset.applications` — A list of one or more resource groups to assign to the Identity Manager account.
- `waveset.organization` — The organization name in which to place the Identity Manager 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,a
ccounts[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 Identity Manager users
- Detect accounts that are not associated with Identity Manager 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 *Identity Manager Resources 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 Identity Manager user that claims the resource to recalculate ownership.

An Identity Manager 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 Identity Manager owner recorded in the Account Index still exists and still claims the account. Any account that does not have an owner is correlated with Identity Manager 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* for 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* 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 Identity Manager owner is correctly recorded, allows incremental reconciliation to skip or shorten several processing phases.

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

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 *Identity Manager Administration*.

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 Identity Manager) 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

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

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

A *correlation rule* looks for Identity Manager users that might own an account. A *confirmation rule* tests an Identity Manager user against an account to determine whether the user actually does own the account. This two-stage approach allows Identity Manager 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.

Identity Manager 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.

Identity Manager 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 Identity Manager 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 Identity Manager 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 Identity Manager 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 Identity Manager 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 Identity Manager 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

Identity Manager 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 Identity Manager 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 Identity Manager 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.

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 *Identity Manager Workflows, Forms, and Views*.

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 Identity Manager 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 Identity Manager 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 Identity Manager 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 Identity Manager. 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 Identity Manager.
- **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 Identity Manager** 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 Identity Manager 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 Identity Manager user. Therefore, it should not be necessary to attempt to manually clear the Account Index for a resource.

Each Identity Manager server runs reconciliation as a daemon task. This means that the Identity Manager 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.

Code 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 Identity Manager IDE. For more details about the Active Sync wizard, see *Identity Manager Administration*.

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 Identity Manager. When the adapter detects that a resource has changed, the Active Sync-enabled adapter:

1. Extracts the changed information from the resource.
2. Determines which Identity Manager object is affected.
3. Builds a map of user attributes to pass to the 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 Identity Manager system uses to update the identity. Typically, a workflow also handles errors within Identity Manager, often ending up as an Approval for a managing administrator.

Active Sync Namespace

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

Table 3-4 Active Sync Processes/Tasks

Process or Task Running	How it is Used	Namespace
ActiveSync `IAPIUser`	• Processes user-related changes on a particular resource. • Performs actions directly on the full User view before launching the designated workflow process.	Merges attributes from the ActiveSync event into the User view. Typical attributes on the Input Form include: • `accounts[*].*` • `waveset.*` • `accountInfo.*` • `activeSync.`<*LHS Attr Name*> • `activeSync.`*resourceName* • `activeSync.resourceId` • `activeSync.resource` • `display.session` (session for Proxy Admin) • `global.`<*LHS Attr Name*> (if `set globals` flag is set on resource)
ActiveSync `IAPIProcess`	• Processes generic events on a resource by creating a Process view. • Top-level fields in Process view are arbitrary inputs to the task. • Collects attributes related to launching the task under the `global` attribute. • Writes the workflow to retrieve inputs from under `global` rather than as top-level attributes.	Launches the specified task with ActiveSync poll attributes dumped into top-level workflow `global` attribute. Workflow attributes assume the form: `global.`<*LHS Attr Name*>

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 Identity Manager user, or creates an Identity Manager 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 3-5 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.
Correlation Rule	If no Identity Manager user's resource info is determined to own the resource account, Identity Manager 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 Identity Manager account: • Identity Manager 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 Identity Manager ID or the user name of an Identity Manager account) If more than one Identity Manager account can be identified by the correlation rule, 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.

Table 3-5 Rule Types *(Continued)*

Parameter	Description
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 Identity Manager identity and the resource account information (placed under the "`account.`" namespace) are passed to the confirmation rule. The confirmation rule is then expected to return a value 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.
Create Unmatched Accounts	If set to true, creates an account on the resource when no matching Identity Manager user is found. If false, Identity Manager 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 Identity Manager cannot find a match with an existing Identity Manager 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 Identity Manager, and creates the user in Identity Manager.

Code 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 *resource 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 resource 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 Identity Manager user when a deletion is detected on the resource.

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

Resource Form

The *resource 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.

Resource 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 Identity Manager 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.

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

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

User Form

The *user form* is used for editing from the Identity Manager 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 Identity Manager 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 *resource 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 resource 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 resource 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 Identity Manager 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 3-6 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.

Code 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 -->
```

Code Example 3-4 Disabling Accounts for Incoming, Inactive Status Attribute *(Continued)*

```
      <Field name='waveset.accountId'>
         <Disable><neq><ref>feedOp</ref><s>create</s></neq></Disable>
         <Expansion>
            <ref>activeSync.EMPLID</ref>
         </Expansion>
      </Field>

    <!-- 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'>
```

Code Example 3-4 Disabling Accounts for Incoming, Inactive Status Attribute *(Continued)*

```
         <Expansion>
            <s>true</s>
         </Expansion>
      </Field>
      <FieldLoop for='name' in='waveset.accounts[*].name'>
      <Field name='accounts[$(name)].disable'>
         <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>
```

Code Example 3-4 Disabling Accounts for Incoming, Inactive Status Attribute *(Continued)*

```
   </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 Identity Manager. 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 Identity Manager, 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 Identity Manager 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 Identity Manager 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 Identity Manager users to departmental administrators or end-users.

Choosing the First Resource

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

- **References a comprehensive set of users**. The goal of an initial load is get as many accounts into Identity Manager 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 Identity Manager account ID**. Each Identity Manager account ID must be unique. Ideally, your resource will have an attribute that is guaranteed to be unique and can be used as a Identity Manager 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 Identity Manager account will be generated.

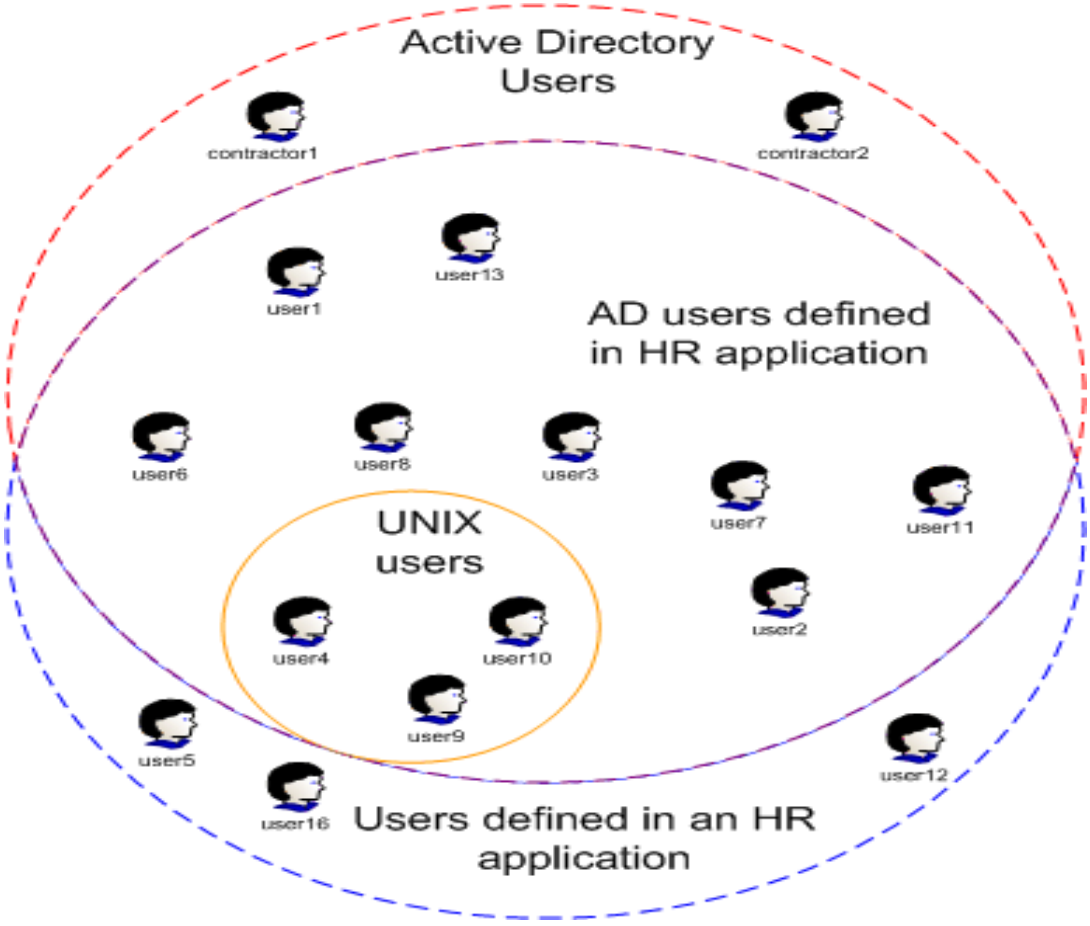
NOTE The Lighthouse account is now called the *Identity Manager* account. You can override this name change through the use of a custom catalog.

See Appendix B, "Enabling Internationalization" for information about custom catalogs.

- **Stores user attributes that can be used correlation keys.** To link resource accounts in Identity Manager, 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 Identity Manager 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 Identity Manager. 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.
- Identity Manager login names. If one resource creates a more desirable login name of Identity Manager, 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 Identity Manager, 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.

Table 4-1 Overview of Data Loading Processes

Data Loading Process	Advantages	Disadvantages
Create bulk action	Allows you to add multiple accounts simultaneously to an Identity Manager user.	• Slower than loading from resource or reconciliation. • Cannot easily generate the CSV file from resources. • Requires detailed knowledge of Identity Manager 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.	

Load from File

The Load from File process seeds Identity Manager 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 Identity Manager. 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 Identity Manager 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 Identity Manager, 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 Identity Manager 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 Identity Manager user. User forms are discussed briefly in Assigning User Forms.

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

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

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 Identity Manager. 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 for more information.

When you seed Identity Manager 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 Identity Manager account index. Therefore, you must perform a full reconciliation, or update the users, before your Identity Manager deployment is complete. In addition, the Load from File does not run any workflows when creating users in Identity Manager.

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 Identity Manager-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 Identity Manager 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,a
ccounts[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,a
ccounts[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,a
ccounts[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 Identity Manager account index. Therefore, you must perform a full reconciliation before your Identity Manager deployment is complete.

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

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 Identity Manager:

- 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 Identity Manager.

For general information about configuring an adapter, see *Identity Manager Administration*. For detailed information about a specific adapter, refer to the *Identity Manager Resources Reference* or the online help.

Setting Account ID and Password Policies

When you load account data from a resource via Load from Resource, reconciliation, or Active Sync, Identity Manager 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 Identity Manager account passwords will not be the same as the those on the resource. By default, Identity Manager 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 Identity Manager account ID. See Assigning User Forms and the chapter titled *Identity Manager Views* 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 Identity Manager accounts, and are categorized as:

- Identity Manager account policies -- Use these to establish user, password, and authentication policy options. Identity Manager 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 Identity Manager.

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

Table 4-2 Default Identity Manager 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 Identity Manager, 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.

Table 4-2 Default Identity Manager Policies *(Continued)*

Policy Name	Default Characteristics
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 the chapter titled *Identity Manager Users* in *Identity Manager Administration* 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 Identity Manager administrator is assigned a user form that creates and edits Identity Manager 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 *Identity Manager Administration* 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 Identity Manager 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 Identity Manager user. It includes:

- Attributes stored in the Identity Manager 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 Identity Manager 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 Identity Manager. The `EmployeeId` attribute is a customized attribute. Its use is illustrated in *Defining Custom Correlation Keys*.

Identity Manager 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 *Identity Manager Forms* for more information about creating and editing forms.

NOTE Do not directly modify a form provided with Identity Manager. 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

Identity Manager uses correlation and confirmation rules to link accounts. A correlation rule looks for Identity Manager 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 Identity Manager 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 Identity Manager 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 Identity Manager `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 Identity Manager.

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 vs. Mary B. Jones, or John Doe Jr. vs. 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 Identity Manager.
- 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 Identity Manager 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:

1. Access the Identity Manager 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 Identity Manager attribute on the Account Attributes (schema map) page for the resource.
6. Save your changes. Identity Manager returns to the System Settings debug page.

The custom attribute must also be added to the `QueryableAttrNames` element in the `UserUIConfig` configuration object.

1. Select Configuration from the **List Objects** pull-down menu. The **List Objects of type: Configuration** page is displayed.
2. Select the **edit** link for **UserUIConfig**.
3. Add the new attributes to the `<QueryableAttrNames><List>` element, for example:

   ```
   <String>EmployeeId</String>
   ```

4. Save your changes. Identity Manager returns to the System Settings debug page.
5. Restart your application server.

Creating Custom Rules

Identity Manager 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 Identity Manager. 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 Identity Manager 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 Identity Manager by using the Identity Manager IDE, importing an XML file, or editing and renaming an existing rule via the debug page.

Manually Linking Accounts

Identity Manager 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 Identity Manager. It is primarily maintained by reconciliation, but other Identity Manager 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_"), Identity Manager displays a menu that presents you with the options of creating a new Identity Manager 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, Identity Manager displays a screen that allows you to search for owners that might criteria that you specify. Refer to *Identity Manager Administration* for more information.

Enabling Self-Discovery

The Identity Manager User Interface can be configured to allow Identity Manager users to discover their own resource accounts. This means that a user with an Identity Manager 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 do this:

1. Access the Identity Manager 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. Identity Manager returns to the System Settings debug page.

When self-discovery is enabled, the user is presented with a new menu item on the Identity Manager User Interface (**Inform Identity Manager 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 Identity Manager 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 Identity Manager 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 Identity Manager 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 Identity Manager.

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` Identity Manager user attribute from schema map. Also make sure the identity template is correct. See the online help and the *Identity Manager Resources 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 Identity Manager 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 Identity Manager users. Since this is the first resource, you probably want to assign the UNMATCHED situation to the value "Create new Identity Manager 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 Identity Manager account policy and default Active Directory identity template, Identity Manager will not create an Identity Manager 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.

Identity Manager 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 Identity Manager 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, with the following modifications:

- When you are configuring the SecurID adapter, ensure that you do not delete the `accountId` Identity Manager 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."

Results

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 Identity Manager `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, with the following modifications:

- When you are configuring the Solaris adapter, ensure that you do not delete the `accountId` or Description Identity Manager 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 Identity Manager. 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.

Results

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
Theodore Benjamin	Theodore Benjamin	tbenjami	Ted Benjamin

In this example, we can expect that only accounts for Isabelle Moreno will 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 Identity Manager 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, we'll assume the e-mail attribute is 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 Employee Number	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 Identity Manager.

1. From the Resources page in the Identity Manager 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 *Identity Manager Resources Reference* for more information.

2. Configure the adapter. Make sure you do not delete the `accountId` or `fullname` Identity Manager 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

Identity Manager 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 Identity Manager 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 Identity Manager User Attribute side of the schema map, and `employeeNumber` to the Resource User Attribute side.

NOTE The PeopleSoft adapter uses the Identity Manager 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 Identity Manager User Attribute side of the schema map, and `mail` to the Resource User Attribute side. `email` is a predefined Identity Manager 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.

Identity Manager 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:

Code Example 4-1 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>
```

NOTE 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, perform the following procedure:

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 Identity Manager User attributes.

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

   c. Ensure that the identity template is correct.

      See the online help and the *Identity Manager Resources 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 Identity Manager user based on resource account" option, the Identity Manager user will have, by default, an account name based on the LDAP `cn` attribute.

3. Reconcile the LDAP resource.

Results

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 Identity Manager User attributes:

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

The USER_EMAIL_MATCHES_ACCOUNT_EMAIL_CORR correlation rule will link the Remedy accounts to the Identity Manager accounts.

To load Remedy accounts, perform the following procedure:

1. From the Resources page in the Identity Manager 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` Identity Manager User attributes. Other attributes can also be added.

   See the online help and the *Identity Manager Resources 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.

3. Reconcile the Remedy resource.

Results

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 Identity Manager using Create actions. Any resources referenced in the CSV file must be defined within Identity Manager before the CSV file is loaded into the system.

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 Lighthouse 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 Identity Manager account that will be used to load accounts into Identity Manager. Assign the user form created in Step 4 to this user.
6. Log in to Identity Manager 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?

Identity Manager 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. Identity Manager 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, Identity Manager 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 Identity Manager 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 Identity Manager, including the account and workflow data necessary to answer questions like those listed above. Identity Manager 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 Identity Manager. 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 Identity Manager.

Exportable Data Types

Data Exporter can export both persistent and transient data. Persistent refers to the data Identity Manager stores in the repository. Transient data is data that is either not stored in the Identity Manager 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 Identity Manager at times that are not externally predictable.

Identity Manager 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
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.

Table 5-1 Supported Data Types *(Continued)*

Data Type	Persistence	Description
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 Identity Manager
TaskInstance	Transient and persistent	A record indicating an executing or completed process
User	Persistent	A logical user that includes zero or more accounts.
WorkflowActivity	Transient	A single activity of an Identity Manager workflow
WorkItem	Transient and persistent	A manual action from an Identity Manager 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, Identity Manager 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. Identity Manager provides SQL scripts that define these tables for each type of supported database. You do not need to modify these tables, unless your Identity Manager 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 96.

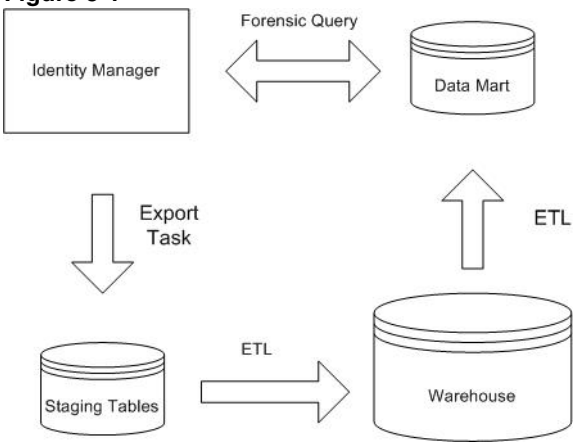
Exporting data to staging tables allows you to write your own Extract, Transform, and Load (ETL) infrastructure so that the data can be processed for storage in a data warehouse, and ultimately, in a datamart. Timestamp manipulation is a commonly-implemented transformation. The system uses the `java.sql.Timestamp` 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 *Identity Manager Administration* for information about configuring Data Exporter.

Forensic queries allow Identity Manager 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 *Identity Manager Administration* for information about defining forensic query.

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

Figure 5-1



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 90 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 92 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 96 for more information.

Database Considerations

Data Exporter can export to any database that is supported as an Identity Manager 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 Identity Manager Java objects and RDBMS tables. Identity Manager 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 96 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

Identity Manager 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, Identity Manager 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, Identity Manager needs a slightly richer expression — and to resolve these cases Identity Manager 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 Identity Manager 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;
}
```

Identity Manager 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 Identity Manager 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 Identity Manager 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, Identity Manager 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 Identity Manager 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 Identity Manager 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 Identity Manager 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 (Identity Manager 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.

Identity Manager provides JMX MBeans to determine the input and output data rates. See *Identity Manager Administration* 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 Identity Manager to match the current export schema.

The `create_warehouse` scripts are located in the `$WSHOME/exporter` directory. Identity Manager 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
```

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 Identity Manager. 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 Identity Manager 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.

Identity Manager 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 Identity Manager servers to provide access to the data objects themselves. This schema is compiled into Identity Manager and represents the data that is stored and operated on within Identity Manager.

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

Custom User and Role attributes, also known as extended attributes, are defined in the `IDMSchemaConfiguration` object. See "Editing Configuration Objects" on page 139 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 Identity Manager 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.
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 Identity Manager. 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

Identity Manager 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 Identity Manager
- Java 1.5 SDK
- ant 1.7 (or later)
- A text editor

The steps required to export extended attributes are as follows:

1. Get the WIC source code from the REF kit
2. Set the `WSHOME` environment variable to the installation directory of Identity Manager
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. By default, these strings are in English. Use the following procedure to display the strings in another language:

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. For example, if you translate into German, the file name should be `WICMessages_de_DE.properties`.

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

Troubleshooting

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 Identity Manager. To minimize the performance impact of exporting data, Identity Manager 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

Configuring User Actions

This chapter details how to add custom tasks to the Identity Manager 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.

Code Example 6-1 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 Identity Manager repository.

Code Example 6-2 Creating an AdminGroup

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

NOTE 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, or
- 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).

Code Example 6-3 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>
```

Code Example 6-3 Changing the Organization for Multiple Users *(Continued)*

```
<?xml version='1.0' encoding='UTF-8'?>
                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>
                   <Transition to='end'/>
                 </Activity>

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

About the Example

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 6-1 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.

Code Example 6-4 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>
```

Code Example 6-4 Changing a User's Organization *(Continued)*

```
        <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.

Generally, the steps for configuring user actions are as follows:

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 Identity Manager 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.

Table 6-2 User Actions Configuration Attributes

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.

Code Example 6-5 Adding 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>
```

Code Example 6-5 Adding Change Organization Task to Each List *(Continued)*

```
            </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>
```

Code Example 6-5 Adding Change Organization Task to Each List *(Continued)*

```
              <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.

Table 6-3 Supported 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.

Table 6-4 userApplet User Action Supported Attributes

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 7

Private Labeling of Identity Manager

This chapter identifies the basic components you will need to rebrand the Identity Manager 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 three 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 Identity Manager 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 7-1 Customizable Components

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

`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 Identity Manager logo
- Using corporate internal look and feel guidelines, specifically borders, header, and background colors
- Changing “logged in as ...” txt. You cannot change this through .jsps. You can edit the custom message catalog. See “Changing Default Information Displayed in the Identity Manager User Interface Home Page” on page 123.

Customizing Identity Manager Pages

Typical customizations include:

- Customizing the home page
- Changing Default Strings in the Identity Manager User Interface Home Page

Customizing the Home Page

- Adding a list of quick links
- Changing the default login text
- Changing page title and subtitle
- Customizing the browser title bar

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 list of 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

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 120.

Code Example 7-1 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 Java&#8482; System [PRODUCT_NAME] system.
Enter the requested information, and then click <b>Login</b>.
UI_LOGIN_WELCOME2=Welcome to the Sun Java&#8482; System [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.

1. Import the following XML file:

Code Example 7-2 XML to Import

```
<?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 Identity Manager User Interface Home Page

The Identity Manager 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 7-2 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 Identity Manager 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 Identity Manager IDE. For information on using the Identity Manager IDE, see *Using the Identity Manager IDE* in *Identity Manager Deployment Tools*. For general information about the System Configuration object, see Appendix A, "Editing Configuration Objects."

Changing the Appearance of the User Interface Navigation Menus

The navigation menus of the User Interface requires two settings in the system configuration object:

- `ui.web.user.showMenu` must be set to `true`
- When `ui.web.user.menuLayout` is set to `horizontal` (default) renders horizontal tabs for the navigation menu. A value of `vertical` renders a menu in a vertical tree menu.

Changing Font Characteristics

Display attributes typically specify the following basic font display characteristics:

Table 7-3 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

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 Identity Manager 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 Identity Manager behavior on commonly used pages

Replacing the Identity Manager 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 Identity Manager 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 look-and-feel of Identity Manager,

1. Edit the `styles/customStyle.css` file.
2. Copy the following sections from `styles.css` into `customStyle.css` and modify as appropriate.

Code Example 7-3 `styles/customStyle.css` File

```
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

Customizing the Identity Manager User Interface Navigation Bar

The Identity Manager 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 Identity Manager 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.

Code Example 7-4 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 {
```

Code Example 7-4 Customizing Navigation Links *(Continued)*

```
    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

Code Example 7-5 Changing Tab Panel Tabs Identity Manager

```
.TabLvl2Div {
    background-image:url(../images/other/dot.gif);
    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;
```

Code Example 7-5 Changing Tab Panel Tabs Identity Manager*(Continued)*

```
    border-top:solid 1px #aba1b5;

table.Tab2TblNew td
{background-image:url(../images/other/dot.gif);background-repeat:repeat-x;background-positi
on: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

Code Example 7-6 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;
```

Code Example 7-6 Changing User / Resource Table Component *(Continued)*

```
    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;
    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 Identity Manager Behavior on Commonly Used Pages

To customize Identity Manager behavior on commonly used pages, you can alter settings in the System Configuration object.

Customizing with the System Configuration Object

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 Identity Manager.

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.

Code Example 7-7 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 7-4 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`).

Code Example 7-8 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 8

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 Identity Manager Retrieves Message Catalog Entries

Identity Manager 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 < and > instead symbols such as < and >.

Message text can also contain variables for data that Identity Manager 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.

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 Identity Manager.
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

The following example creates a customized message catalog named `myCustomCatalog`. It replaces the label and help text for the **Import Exchange File** subtab.

Code Example 8-1 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>
```

Appendix A

Editing Configuration Objects

This chapter introduces an Identity Manager component called a *configuration object*. Configuration objects store persistent customizations to Identity Manager. 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 Identity Manager repository are configuration objects.

Editing configuration object properties is one way of implementing persistent changes to Identity Manager behavior. This appendix describes how to view and edit configuration objects. The information is organized as follows:

Data Storage

Sun Identity Manager 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 Identity Manager 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 Identity Manager organizational information.
- **userobj** – Stores all the Identity Manager 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 Identity Manager 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). Identity Manager stores data in this way because all Identity Manager 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 UserView, PasswordView, etc.).

Object Naming Conventions

Do not use the following characters in any Identity Manager object names:

Character	Description
'	single quotation mark
=	equal sign
.	period
|	vertical bar
[	left bracket
]	right bright
,	comma
:	colon
$	dollar sign
\	backslash
"	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 Identity Manager behavior.

You can use the Sun Identity Manager Integrated Development Environment (Identity Manager IDE) to view and edit Identity Manager objects for your deployment. Instructions for installing and configuring the Identity Manager Integrated Development Environment (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
- UserUIConfig Object
- RepositoryConfiguration Object
- WorkItemTypes Configuration Object
- SystemConfiguration Object
- Role Configuration Object
- End User Tasks Object

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>
```

NOTE 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 154.

`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.

Identity Manager 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 Identity Manager 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 multivalued, 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.

Figure A-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'
```

Figure A-1 Extended Attributes Example *(Continued)*

```
                                                    summary='true'/>Configuration
list/read/write.
                  <IDMObjectClassAttributeConfiguration name='lastname'
                                                    queryable='true'
                                                    summary='true'/>
                  <IDMObjectClassAttributeConfiguration name='fullname'/>
...
              </IDMObjectClassConfiguration>
          </IDMObjectClassConfigurations>
       </IDMSchemaConfiguration>
   </Configuration>
</Waveset>
```

NOTE *To prevent potential conflicts with new core attributes in future releases of Sun Identity Manager*, 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 Identity Manager incorporates a built-in user attribute named `employeeNumber`, the two attributes will remain distinct. Otherwise the built-in attribute takes precedence.

Extending the Role ObjectClass

You can extend Role using an `IDMObjectClassConfiguration`. The following built-in Role extensions all extend the Role objectclass:

- `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 150.

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."

SummaryAttrRoleCountLimit

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, Identity Manager will list at most three roles.

RepositoryConfiguration Object

The RepositoryConfiguration object contains settings that control the behavior of the Identity Manager 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 154.

NOTE Changes to the `RepositoryConfiguration` object do not take effect until you restart each Identity Manager server. Restarting an Identity Manager 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 Identity Manager Tuning, Troubleshooting, and Error Messages 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 Identity Manager 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.

t

Table A-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
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, we do not document the full range of possible customizations here. Some common customizations are documented here.

Controlling the Display of the Password Confirmation Popup

The forgotPasswordChangeResults attribute in the System Configuration object controls whether Identity Manager 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.

Registering Scheduler Startup (for Clustered Environments)

The scheduler.hosts attribute registers startup behavior for the scheduler for each Identity Manager 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 Identity Manager application instance.

NOTE To see the hostname value, go to the debug/GetStatus.jsp page in your Identity Manager 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>
```

`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
- Actions
- List Columns

Types

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`
- `authType`
- `workItemTypes`
- `features`

`displayName`

Specifies the type's display name whose value is a message catalog key.

`authType`

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`

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`

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 ResourceGroups 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

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

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 Identity Manager 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 Identity Manager adds a new attribute, or when a new version of Identity Manager 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 reserialize 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 reserialized.

- Use the `lh refreshType` utility to reserialize 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, reserializing 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 Integrated Development Environment (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.

Appendix B

Enabling Internationalization

This appendix describes how to configure Identity Manager to use multiple languages or to display a language other than English.

Architectural Overview

Table B-1 Components of Identity Manager Internationalization

File	Description
`WPMessages.properties`	Default message file located in `$WSHOME/idm/web/WEB-INF/classes/com/waveset/msgcat`. Shipped as part of the `idmcommon.jar` file. Displays message text in English and loads by default unless you've customized your Identity Manager installation to behave otherwise.
`Waveset.properties`	Located in `$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=Identity Manager
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 Identity Manager.

Step One: Download and Install Localized Files

Before You Install

Perform the following tasks before you install localized files:

1. Install Identity Manager. See *Identity Manager Installation* 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 Identity Manager 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 B-2 Message Catalog Files

File Name (.zip)	Language	Locale
IDM__8_0_l10n_de	German	de_DE
IDM__8_0_l10n_es	Spanish	es_ES
IDM__8_0_l10n_fr	French (France and Canada)	fr_FR
IDM__8_0_l10n_it	Italian	it_IT
IDM__8_0_l10n_ja	Japanese	ja_JP
IDM__8_0_l10n_ko	Korean	ko_KR
IDM__8_0_l10n_pt	Brazilian Portuguese	pt_BR
IDM__8_0_l10n_zh	Simplified Chinese	zh_CN
IDM__8_0_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_5_0_l10n_Locale.jar`.
- Identity Manager Localization README

Additional translated publications might also be available.

Install Localized Files

Use the following steps to install localized files on your application server.

1. 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,

1. Open the `IdentityManagerInstallation/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 Identity Manager 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, Identity Manager'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 Identity Manager 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:

1. Edit the following rules in the EndUserRuleLibrary as indicated in the following table:

Table B-3 Editing the EndUserRuleLibrary rules

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

D

E

F

H

I

J

L

M

O

P

Q

R

S

T

U

V

W

X