Sun Identity Manager Service Provider 8.1 Deployment

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

Part No: 820–5601

090312@21990

Contents

Preface

Identity Manager Service Provider Deployment provides reference and procedural information that describes how to plan and implement Sun Java™ System Identity Manager Service Provider..

Who Should Use This Book

Sun Java System Identity Manager Service Provider Deployment was designed for deployment engineers who will deploy Service Provider.

Before You Read This Book

Deploying this product requires knowledge about Identity Manager, LDAP directories, and optionally, federation management. For a broader discussion about Service Provider, see *Sun Identity Manager 8.1 Business Administrator's Guide*. See *Sun Identity Manager Deployment Guide* for more information about implementing Identity Manager.

How This Book Is Organized

Identity Manager Service Provider Deployment is organized into these chapters:

- Chapter 1, "Identity Manager Service Provider Overview"— Describes key concepts and components of the product.
- Chapter 2, "Planning the Service Provider Installation"— Discusses issues that need to be considered before you install and implement Service Provider.
- Chapter 3, "LighthouseContext API"— Describes the Service Provider application program interface (API) and persistent objects.
- Chapter 4, "IDMXUser View"— Provides detailed information about the IDMXUser view, and describes how it varies from Identity Manager views.
- Chapter 5, "Other Objects in Service Provider"— Describes how user forms and resources differ in Service Provider.
- Chapter 6, "Service Provider User Interface"— Describes the sample User Interface pages provided with the system.

- Chapter 7, "Implementing Custom User Pages"— Provides detailed information about deploying a set of user pages.

Related Books

The Sun Identity Manager 8.1 documentation set includes the following books.

Primary Audience	Title	Description
All Audiences	*Sun Identity Manager Overview*	Provides an overview of Identity Manager features and functionality. Provides product architecture information and describes how Identity Manager integrates with other Sun products, such as Sun Open SSO Enterprise and Sun Role Manager.
	Sun Identity Manager 8.1 Release Notes	Describes known issues, fixed issues, and late-breaking information not already provided in the Identity Manager documentation set.
System Administrators	*Installation Guide*	Describes how to install Identity Manager and optional components such as the Sun Identity Manager Gateway and PasswordSync.
	Upgrade Guide	Provides instructions on how to upgrade from an older version of Identity Manager to a newer version.
	System Administrator's Guide	Contains information and instructions to help system administrators manage, tune, and troubleshoot their Identity Manager installation.
Business Administrators	*Business Administrator's Guide*	Describes how to use Identity Manager provisioning and auditing features. Contains information about the user interfaces, user and account management, reporting, and more.

Primary Audience	Title	Description
System Integrators	*Deployment Guide*	Describes how to deploy Identity Manager in complex IT environments. Topics covered include working with identity attributes, data loading and synchronization, configuring user actions, applying custom branding, and so on.
	Deployment Reference	Contains information about workflows, forms, views, and rules, as well as the XPRESS language.
	Resources Reference	Provides information about installing, configuring, and using resource adapters.
	Service Provider 8.1 Deployment	Describes how to deploy Sun Identity Manager Service Provider, and how views, forms, and resources differ from the standard Identity Manager product.
	Web Services Guide	Describes how to configure SPML support, which SPML features are supported (and why), and how to extend support in the field.

Documentation Updates

Corrections and updates to this and other Sun Identity Manager publications are posted to the Identity Manager Documentation Updates website:

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

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

Related Third-Party Web Site References

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

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

Documentation, Support, and Training

The Sun web site provides information about the following additional resources:

- Documentation (`http://www.sun.com/documentation/`)
- Support (`http://www.sun.com/support/`)
- Training (`http://www.sun.com/training/`)

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 Feedback.

Typographic Conventions

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

TABLE P–1 Typographic Conventions

Typeface	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename.*

TABLE P–1 Typographic Conventions *(Continued)*

Typeface	Meaning	Example
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

The following table shows the default UNIX® system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–2 Shell Prompts

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

◆ ◆ ◆ CHAPTER 1

Identity Manager Service Provider Overview

This chapter provides an overview of the information that an engineer needs to know to deploy Sun Java™ System Identity Manager Service Provider. Deploying this product requires knowledge about Identity Manager, LDAP directories, and optionally, federation management. For a broader discussion about Service Provider, see Business Administrator's Guide. See Deployment Guide for more information about implementing Identity Manager.

What Is Identity Manager Service Provider?

Identity Manager Service Provider is a highly-scalable, **extranet** -focused identity management solution that is capable of provisioning and maintaining millions of end-user accounts that are stored on an LDAP directory server. Service Provider can also manages thousands of administrator accounts and synchronizes LDAP account data with other resources.

Service Provider is a component of Identity Manager and is installed automatically. However, deploying Service Provider functionality requires additional planning and effort beyond that required for Identity Manager.

When Should Service Provider Be Deployed?

You need to manage millions of extranet accounts defined in an LDAP directory.

- You should deploy Service Provider in the following circumstances:
- You have any number of employee and other internal accounts defined in an LDAP directory that can be managed simply.
- You do not need the complex capabilities that Identity Manager workflows provide.

Service Provider Features

An Service Provider deployment adds the following to Identity Manager:

- A **user view** that is specific to Service Provider users. The user view provides a data model for describing the provisioning operations to be performed, including attribute modifications, password resets, and account disables.
- **Transaction manager**. Provides a mechanism for executing provisioning requests and ensures all resource operations are performed, including those that need to be retried due to resource or server failure. Transactions are logged in a separate database from the Identity Manager repository.
- **Enhanced reporting capabilities**. Service Provider tracks system events, such as concurrent users and administrators, resource operations, and operation failures. Administrators can use the dashboard graph feature on the Identity Manager Administrator interface to quickly assess the current system and spot abnormalities, and to understand historical trends (such as concurrent users or resource operations over a time period.) For more information about this feature, see Business Administrator's Guide.

Service Provider also provides the following features:

- **Delegated administration**. Administrators can view and edit only users that they control. This is enforced by assigning organizations and Service Provider-specific capabilities to administrators, or dynamically by rules granting finer grained capabilities and rights.
- **Synchronization services**. Service Provider also synchronizes account information on the LDAP directory with accounts on other resources.

Service Provider does not have its own Administrator Interface. All administration tasks, such as system configuration and viewing dashboard graphs are performed from the Identity Manager Administration Interface. Service Provider provides a set of sample User pages that illustrate how the product can be implemented, but customizations are required for these pages.

Differences Between Identity Manager and Service Provider

Service Provider does not use many of the features present in Identity Manager, because they are less useful in large-scale “service provider” environments. Concepts not used in Service Provider include:

- Reconciliation and loading from resources
- Risk analysis
- Compliance

Some concepts are applicable to both products, including the following:

- Resource and Active Sync adapters
- Forms
- Delegated administration
- Roles and rules
- SPML
- Infrastructure code, such as tracing and utilities

The following sections describe some of the differences in detail.

IDMXUser View

The IDMXUser view is similar to the Identity Manager User View. Both views allow the caller to create or check out a view, make changes to the view, and check in the results. However, the attributes within the two views differ greatly.

The IDMXUser view is much narrower in scope than the Identity Manager User view. For example, the IDMXUser view does not contain the global or password top-level attributes. The waveset attribute is not supported in IDMXUser, but some of its sub-attributes are supported through other attributes in IDMXUser. The IDMXUser view does not return resource-specific attributes unless specifically requested.

For a full description of the IDMXUser view and a comparison between the IDMXUser view and the Identity Manager User view, see Chapter 4, "IDMXUser View."

Repository

Service Provider bypasses the Identity Manager repository in the following ways:

- User accounts are not loaded into the repository. Instead, all account information is stored in an LDAP directory. Information that would be in the account index in Identity Manager is stored in a configurable LDAP attribute. Alternatively, an auxiliary object class may be created to identify the existing and to-be-provisioned users. However, if there are existing users in the master directory store, then this new auxiliary object class must be retroactively added to these users.
- Transaction information is written to a separate database.

Workflows and Approvals

Workflows are powerful tools for provisioning users and establishing approvals in Identity Manager. However, because workflows often result in complex transactions involving human interaction, they are not ideally suited for environments in which simple provisioning actions need to be performed on millions of users.

As a result, Service Provider does not use workflows. Instead, Service Provider uses a transaction manager to carry out transactions, such as resource operations and updates to

LDAP meta-data. The transaction manager persists all transactions into a database and ensures that any transaction failures caused by a resource or the Service Provider server are completed.

See "Workflow Callouts" on page 55 for information about how callouts provide some of the functionality handled by workflows.

Authentication and Authorization

Service Provider does not require authentication or authorization when performing provisioning actions through the LighthouseContext API. A portal or an access management application can perform these services. Communications between the portal and Service Provider (if using SPML) must be secured by using SSL or similar technology.

Authentication and authorization are performed by Identity Manager. Service Provider administrators can be Identity Manager or Service Provider users that are assigned Service Provider-specific capabilities and are able to control organizations. The organizations are created in Identity Manager, but the administrators that belong in each organization are defined by the customers. These administrators can be determined by searching for specific values on LDAP attributes, or by enabling external authorization. External authorization on Service Provider Users can be enabled for the Service Provider End User interface or Identity Manager Administrator interface.

When enabled, viewing, creating, updating, and deleting can be controlled by one or more Service Provider User AdminRoles assigned to the user (Service Provider or Identity Manager) making the request. Whether the user is allowed to do the action is controlled the evaluation of Rules assigned one or more AdminRoles assigned to the user, which use external resource data to determine whether to grant access or not.

Data Loading Mechanisms

Service Provider does not load LDAP user account information into the repository. Instead, it uses the information already in place in the directory. As a result, there is no need to configure the resource adapter to perform reconciliation; nor is it necessary to perform a load from resource or a load from file operation to populate the account index with end user accounts.

However, if administrator accounts are not already defined within Identity Manager, you might need to populate administrator accounts into the repository. Any data loading mechanism can be used to accomplish this task. See Business Administrator's Guide for more information.

User Interface

Service Provider provides a set of sample end-user pages that can be used as the starting point of your own user interface. The sample end-user pages are implemented using the Apache Struts Tiles Framework. This allows you to easily customize the default pages.

Implementation Overview

To implement Service Provider, you will need to perform the following tasks:

- Install Identity Manager. You must create a transaction database as part of the installation process. See Installation Guide for detailed instructions.
- Configure Service Provider as directed in Business Administrator's Guide. See Chapter 5, "Other Objects in Service Provider," for information about customizing user forms and rules.
- Customize the end user pages or create a portal application that allows end-users to perform actions such as creating accounts, editing attributes, including passwords. The sample end-user pages are a starting point. You must use the LighthouseContext API or SPML to communicate with Service Provider. The IDMXUser view is used to perform all provisioning tasks.

Using the Service Provider REF Kit

Service Provider provides a REF kit to help you implement the product. This kit is located in the `idmspe-refkit.zip` file in the root directory of the installation media. The kit contains the following:

- JAR files needed to develop and deploy Service Provider custom code.
- A sample Java file (`ApiUsage.java`) that implements the most important features of the LighthouseContext and IDMXUser view.
- A sample Java file (SpmlUsage.java) that demonstrates how to use implement Service Provider using SPML.
- Several sample audit handlers, filters, and formatters.
- Authentication/authorization filter used in the sample end-user pages.
- The `build.xml` file, which is an Ant script that can be used to build, deliver, and test source.
- Javadocs

Requirements

The REF kit has the following requirements:

- JDK 1.5
- Ant 1.6.3 or higher

Optional for testing:

- JUnit 3.8.x or higher

You might also need to copy any additional JAR files to the `lib` directory. The build will automatically load the JAR files for both compiling and testing.

See `http://ant.apache.org` for more information about using Ant.

Building with the REF Kit

Use the following general procedure to set up your development and testing environment. Refer to the JDK and Ant documentation for detailed instructions.

▼ Set Up Development and Testing Environment:

1. **Install JDK 1.5 or higher.**
2. **Install Apache Ant 1.6.3 or higher.**
3. **Set the JAVA_HOME environment variable to point to the JDK. For example:**

   ```
   JAVA_HOME=c:\jdk-1.5.0_x
   ```

4. **Set the ANT_HOME environment variable to point to Ant. For example:**

   ```
   ANT_HOME=c:\apache-ant-1.6.3
   ```

5. **Add `JAVA_HOME\bin` and `ANT_HOME\bin` the PATH environment variable.**
6. **Install the REF kit.**

   ```
   mkdir spe-dev
   copy idmspe-refkit.zip spe-dev
   cd spe-dev
   unzip idmspe-refkit.zip
   ```

   You are now ready to change directories to `spe-dev` and run `ant`. You may run `ant` with or without an optional target. To specify a target, enter `ant` *TargetName*, such as `ant compile`. If you do not specify a target, `ant` will build everything in the `src` directory and create a JAR file in a new directory, `dist`.

   The following table lists the possible targets.

Target	Description
init	Initializes the build environment by creating dependent directories.

Target	Description
compile	Compiles all source in the `src` directory into the `build` directory.
resources	Copies all non-Java source files into the `build` directory. This is for `properties` file support and other embedded resources.
dist	Create a JAR file from the files in the `build` directory into the `dist` directory. The name of the JAR will be the project name specified at the top of the `build.xml` file "MyProject" and a timestamp.
clean	Removes the `dist`, `build`, and `reports` directories and remove all `.class` files from the `test` directory.
test	Calls the compile, resources, test-compile, and test-run targets (in that order), then runs the JUnit reports task on the results to produce both HTML and text reports on the success or failure of the tests. The reports are written to the `reports` directory.
test-run	Calls the compile, resources, and test-compile targets. Then it runs each test case within the `test` directory whose file name contains the pattern `*Tests.java`.
test-compile	Calls the compile, and resource targets, then compiles all the files within the `test` directory.

◆ ◆ ◆ CHAPTER 2

Planning the Service Provider Installation

This chapter provides an overview of topics that must be considered before deploying Service Provider. Deploying this product requires knowledge about Identity Manager, LDAP directories, access control applications (such as Sun Java™ System Access Manager), and optionally, federation management. However, the topics of access control and federation management are not discussed.

Determining the Architecture

Service Provider may be deployed in either a one-tier or two-tier architecture. The following sections describe both of these architectures.

All implementations require a relational database for storing transaction information. There are no specific requirements for the database, or its host, other than it must be able to connect to Service Provider with JDBC. See the release notes for a list of supported databases.

Service Provider uses transactions to encapsulate provisioning work. (Workflows are not implemented in this product, but callouts are available.) Transactions include resource operations as well as updates to meta-data in a user's entry in the master directory. The transaction manager, which executes these transactions, is configurable from the Identity Manager Administrator Interface. Configuration options include whether to process transactions synchronously, and when and how often to persist transactions. Transactions are stored in a database, and are tracked to completion through resource or server failures.

Service Provider requires an LDAP directory to query and provision user accounts. LDAP user account information is not stored in an Identity Manager repository. However, information stored in an LDAP directory may be provisioned to other accounts on other resources.

One-Tier Architecture

In a one-tier architecture, Service Provider and the user interface are installed on the same application server (or servers). This option is less secure, because the web server must have access to the internal databases and resources. The following diagram illustrates Service Provider in a one-tier environment.

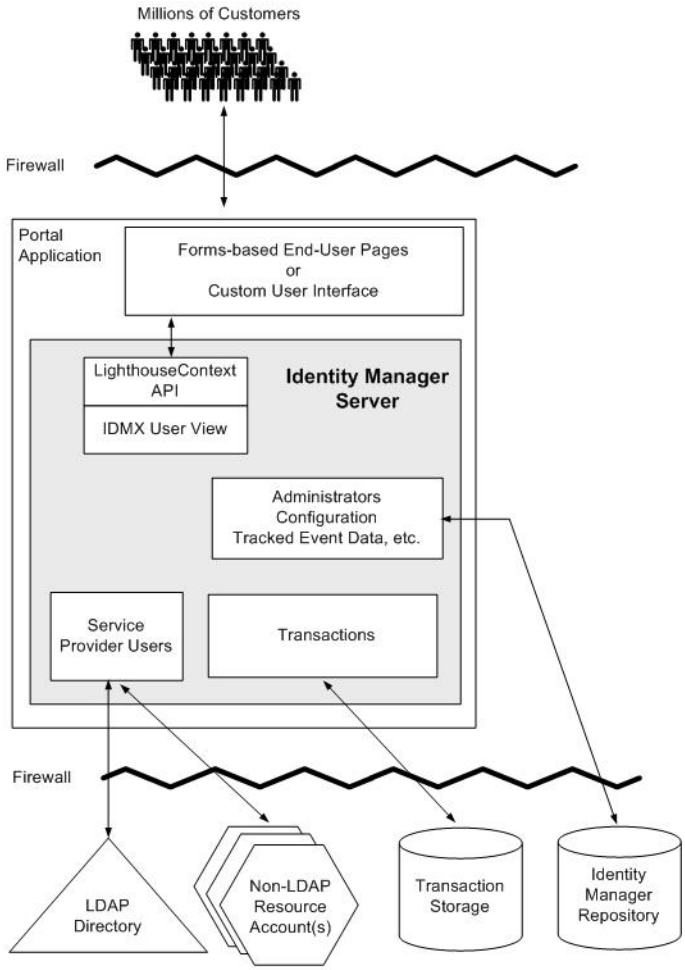

FIGURE 2–1 Service Provider in a One-Tier Architecture

Two-Tier Architecture

In a two-tier architecture, the portal is in a demilitarized zone (DMZ), while Service Provider remains secure within the enterprise. The portal accesses Service Provider over SPML or with a RemoteContext.

Implementing a two-tier architecture means you must take additional security precautions. It is recommended that you perform the following steps to secure your network:

- Install a firewall between the portal server and the Service Provider server.
- Use either HTTPS or HTTP in conjunction with SSL for communications between the servers. This is especially true if Service Provider resides in an untrusted domain.
- Restrict the IP addresses that the portal server and Service Provider server can use to communicate.

The following diagram illustrates how Service Provider can be implemented in a two-tier architecture.

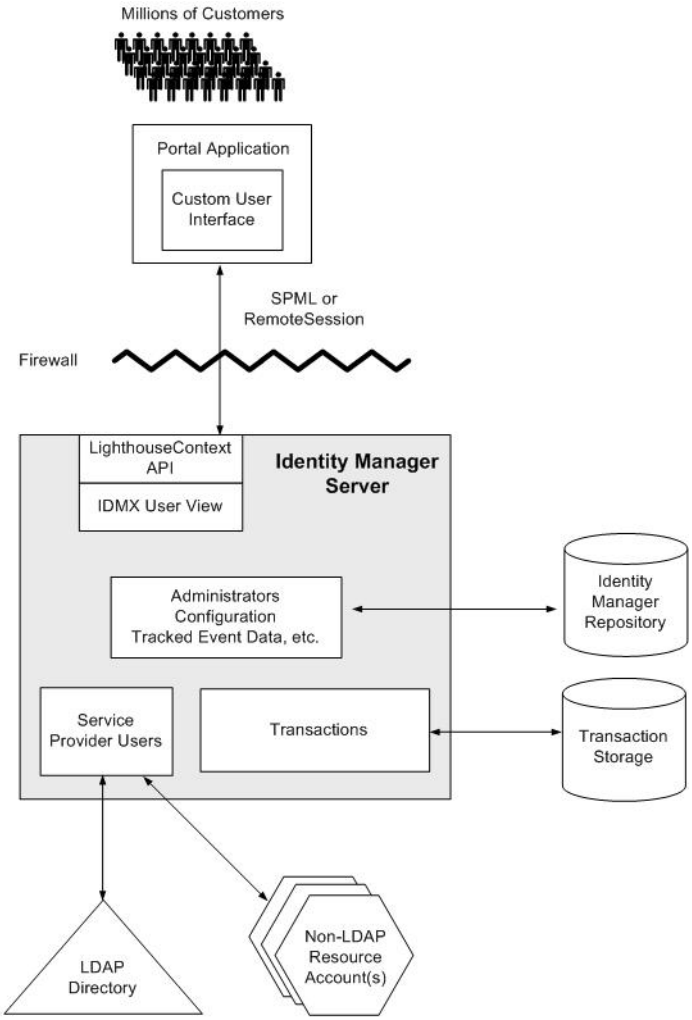

FIGURE 2–2 Two-Tier Architecture with a Custom User Interface

Planning Delegated Administration

Delegated administration of Service Provider users is enabled through the following means:

Organization-Based Authorization

Service Provider administrators are Identity Manager users that are assigned capabilities and can control organizations. These administrators can be created and maintained in the same way as Identity Manager administrators.

Several different levels of administrators might be needed in your environment to perform various tasks, such as the following:

- The lowest level of administrator might be permitted to create and edit end user (customer) accounts only. There might be thousands administrators with these capabilities.
- Another level of administrator might be permitted to create, edit, and delete low-level administrators as well as customer accounts. There might be hundreds of such administrators.
- A set of administrators to manage the LDAP directory and other resources. There might be several dozen of this type of administrator.
- A few high-level administrators that have super user abilities.

The last two categories of administrators would likely be created and maintained manually in the Administrator Interface. However, to create the lower-level administrators, perform the following tasks:

1. Review the object classes and attributes present in your LDAP directory and determine which of these items contain data that indicates the user should have administrative powers. For example, you might have an attribute that indicates a user is a retailer or manager. A retailer would have the ability to create accounts. A manager might be permitted to create retailer accounts.
2. Add the attributes to the schema map of the LDAP resource.
3. Determine how you will limit the scope of each administrator. For example, a retailer based in Texas probably should not be allowed to create accounts for users who live in New York. Similarly, a manager should not be allowed to delete accounts outside her realm.
4. Create organizations in Identity Manager that correspond to the scopes defined in the previous step.
5. Create a user form that creates an Identity Manager account, assigns capabilities as well as an organization for each administrator. The user form must use the attributes defined in "Organization-Based Authorization" on page 26.

6. Use reconciliation or other data loading mechanism to load the administrator accounts into Identity Manager.

Admin Roles

For granting fine-grain capabilities and scope of control on Service Provider users, use an Admin Role whose authType is `ServiceProviderUserAdminRole`. The Admin Roles can be configured to be dynamically assigned to one or more Identity Manager or Service Provider Users at login time.

Rules can be defined and applied to the Admin Roles that specify the capabilities (such as `Service Provider Create User`) of the members of that admin role.

To use Admin Role delegation for Service Provider users, you must enable it in the Identity Manager system configuration object. See Business Administrator's Guide for detailed information about this task.

To define this type of Admin Role, you must create one or more rules. See "Delegated Administration" on page 56 for more information.

◆ ◆ ◆ CHAPTER 3

LighthouseContext API

LighthouseContext is the name of a Java API that is used to configure the Service Provider server and access provisioning services. Customer-developed web applications, which can be deployed on a portal server, use this API.

Refer to the Javadocs and the sample Java file (`ApiUsage.java`) in the REF kit for more details about the LighthouseContext API with Service Provider.

Obtaining a LighthouseContext

There are four varieties of context:

- Local Anonymous
- Local Authenticated
- Remote Anonymous
- Remote Authenticated

A local context calls the Identity Manager classes directly, while remote access can be achieved using SPML. Note that the Service Provider SPML handler does not perform authentication. The authentication should be executed by the portal application.

An anonymous context does not require a user name and password. If auditing is performed, it will use a system defined name. There is an anonymous local context, the *internal context*, that is used in the implementation of the Service Provider server. The internal context may also be accessed by application code, though note that you will lose the ability to record specific end-user names in the audit log.

All varieties of context are obtained from factory methods on the SessionFactory class. The factory methods are:

Connection Type	Method	Description
Local anonymous	getServerInternalContext()	Returns a fully authorized context without any authentication.
Local authenticated	getSPESession(String user, EncryptedData password)	Constructs a session for the Service Provider user interface.
Local authenticated	getSPESession(Map credentials)	Constructs a session for the Service Provider user interface. The map specifies the credentials of the user, including the values of the user and password keys.
Local pre-authenticated	getSPEPreAuthenticatedSession(String user)	Constructs a pre-authenticated session for the Service Provider user interface.
Remote anonymous	Not applicable	This connection type is only available through SPML.
Remote authenticated	getSession(URL url, String user, EncryptedData pass)	Returns an authenticated session.

Persistent Objects and Views

Users in the LDAP directory can be accessed as persistent objects. In addition to retrieving the normal attributes stored in the directory, the context also retrieves extended provisioning information stored in special attributes. This includes information about links to other accounts associated with the directory user. This extended information is only available through the context API, it cannot be accessed by pure LDAP applications.

A second object model called Views may also be accessed through the context. View objects are not stored in the directory, rather they are assembled at runtime from one or more persistent objects. User objects in the directory are most often accessed through a view rather than a persistent object. The user view contains all of the information found in the persistent object, but may in addition contain attribute from other accounts linked to this user. Manipulating a user view is how all provisioning operations are performed in Service Provider.

Views are represented in memory using a generic memory model based on simple Java data types such as lists and hash maps. This regular structure allows views to be easily manipulated by technologies such as forms, and makes the view attributes easier to associate with HTTP form fields. For more information about views, see Chapter 4, “IDMXUser View.”

Persistent Object Identification

Persistent objects are identified by the following pieces of information.

- A `Type` object that specifies a repository storage type.
- A `String` that specifies the object's unique repository ID or repository name.

Service Provider uses the IDMXUser object type. Its corresponding static Type instance is Type.IDMX_USER.

A unique repository ID is generated by the system and may be used programmatically, but it is normally not intelligible to an end-user. Once assigned, the repository ID cannot be changed. The format of the identifier is considered unspecified, and the application should make no assumptions about the characters it contains. Note that the ID might contain fragments of a DN, but this DN does not necessarily match the actual DN of the object.

The repository name is specified by the user. When displaying the name of a persistent object in a UI application, you should always display the name rather than the ID. The object name is guaranteed to be unique among the objects of the same type, but is not necessarily unique among objects of different types. Object names may be changed.

Because object names can change, it is generally more reliable for an application to use repository IDs, if they are available. For interactive applications, this may not be possible if the names are being entered by the user. For this reason, the LighthouseContext methods will accept either an identifier or a name.

Option Maps

Many of the LighthouseContext methods take an argument of type `java.util.Map` called an *option map*. This provides an extensible way to pass additional information into the function rather than through a lengthy list of arguments, or a large collection of different method signatures.

The map keys are always strings. The map values are usually strings or lists of strings. When an option value is treated as a boolean, the value may either be the strings `true` and `false` or a `java.lang.Boolean` object. When an object value is treated as an integer, it may either be a string or a `java.lang.Integer` object.

The options recognized by a particular method will be described in the documentation for that method. The following table lists some common options recognized by many methods.

Option	Description
user	Used to specify an alternate user name for auditing and locking. If you are using an authenticated context, the name of the authenticated user will be used by default. Specifying this option is recommended if you are using a shared anonymous context so that you can trace audit log entries back to a particular user.
allowNotFound	Used by methods that would ordinarily throw the ItemNotFound exception if you attempt to retrieve an object with an invalid name or id. When this boolean option is true, the method will silently return null or ignore the request rather than throwing an exception.
form	Specifies an alternate form to be used to process the view during check-out, refresh, and check-in.

LighthouseContext in Service Provider

This section describes how the LighthouseContext is used differently within Service Provider.

Object Locks

Service Provider does not support object locks.

Do not use the following methods, which are defined in the ObjectSource interface:

- lockobject
- unlockobject
- getLock
- breakLock

Persistent Object Queries

If you are writing a generic repository browsing application, you will need perform searches to discover the objects in the repository. There are two styles of searches, a listing search (`listObjects`) and a fetching search (`getObjects`). A listing search retrieves the name and a few attributes of the object, but does not bring the entire object into memory. A fetching search loads the entire object into memory.

You should always use listing searches when querying IDMXUser objects, because there can be a large number of users in the directory.

The `listObjects`, `getObjects`, and `deleteObjects` methods recognize an options map argument, in which search criteria and other options may be specified. When building an options map, it is recommend that you use a set of constant string objects. The following table lists the valid query option names and their corresponding constant object.

Option Name	Object
orderBy	LighthouseContext.OP_ORDER_BY
maxRows	LighthouseContext.OP_MAX_ROWS
subject	LighthouseContext.OP_SUBJECT
attributes	LighthouseContext.OP_ATTRIBUTES
conditions	LighthouseContext.OP_CONDITIONS

Service Provider enforces a maximum number of results that a query can return. This maximum is specified by the **maxRows** option.

User Objects

Since Service Provider users are also persistent objects, you can create and modify them in the same way as configuration objects. While this is the fastest way to create users in the directory, it is important to understand that the provisioning of resource accounts does not happen when you use the persistent object methods. If you want to perform provisioning you must use the IDMX User View and the view methods.

The `IDMXUser` class is a subclass of `Composite`, which provides a generic model to represent an object with a collection of attributes, assignments to resources, and links to the associated objects on those resources. The structure of the `IDMXUser` class is similar to the Identity Manager `WSUser` class, but the field and method names are slightly different, because the model is intended for use with entities other than user accounts.

The following table shows the correspondence of some of the major classes and field names in the two products.

Description	Identity Manager	Service Provider
General user class	WSUser	Composite
Attribute class	WSAttributes	GenericObject
Add resource method	WSUser.addPrivateResource	Composite.addAssignment
Resource account class	ResourceInfo	Link
Get account identity method	ResourceInfo.getAccountId	Link.getIdentity
Set account attributes method	ResourceInfo.setAttributes	Link.setPendingAttributes

When you set the attributes of an IDMXUser, it is important to know how these attributes are being mapped to attributes in the directory. This mapping is defined by a Resource definition in the repository, which is named Service Provider End-User Directory in the examples.

This resource is available in Identity Manager by default. The resource has a schema map set up to work with the Service Provider example end user pages.

IDMXUser Attribute	Directory Attribute
name	uid
password	userPassword
firstname	givenname
lastname	sn
objectClass	objectClass
fullname	cn
xml	jpegPhoto
email	mail
homephone	telephoneNumber
cellphone	mobile
passwordRetryCount	passwordRetryCount
accountUnlockTime	accountUnlockTime

The full DN of this user will be built according to the identity template defined in the resource. If you want to assign a DN whose structure differs from that in the identity template, call the `setIdentity` method to specify the full DN.

Any attribute that you set on an `IDMXUser` that is not defined in the directory schema map will be stored as an extended attribute. Extended attributes are stored in the XML blob with the other object metadata, they are not directly accessible in the directory.

When you no longer need a persistent object, it may be deleted to reclaim space and reduce clutter in searches. Service Provider does not perform reference checks before deleting an object. This means that it is possible to delete a Resource object, for example, while that object is still being referenced by a Role object.

◆ ◆ ◆ CHAPTER 4

IDMXUser View

The IDMXUser view is similar in structure to the Identity Manager user view, though attribute names have been changed to make it less specific to users and accounts. Instead of referring to the term "user", this discussion will use the terms "composite object" or "master object" The term "linked object" will be used instead of "account."

Refer to the Javadocs and the sample Java file (`ApiUsage.java`) in the REF kit for more details about the IDMXUser view.

Implementing the IDMXUser View

Manipulation of directory users is usually performed through the IDMXUser view rather than with the persistent object methods. The primary difference is that the view supports provisioning operations, while the persistent object methods simply modify the data stored in the directory.

The general process for using a view is as follows:

1. Create or check out a view, which is represented in memory as a `GenericObject`.
2. Make changes to the attributes in the view.
3. Check in the view. The system might need to perform complex actions as a result of the changes, including provisioning accounts and updating the directory.

   See Deployment Guide for more information about implementing views

IDMXUser View Reference

Attributes in the IDMXUser view are divided into the following categories:

- **Account attributes** are values that are actually stored in a resource account or in the directory. They are defined by the schema map of a Resource definition, and are the attributes you most often modify when creating or editing accounts.

  Commonly used account attributes include `accountId`, `password`, `email`, and `fullname`.

- **System attributes** are values defined by the Service Provider system, but are independent of resource type, and not necessarily stored on the resource or in the directory. They are used to request certain operations, or store information that cannot be represented with native account attributes. System attributes are collected under a top-level attribute named `sys` to avoid name conflicts with account attributes.

  Commonly used system attributes include `sys.identity` , `sys.links`, `sys.delete`, and `sys.expirePassword`.

  A subcategory of system attributes are called action attributes. An action attribute is set to the value `true` to indicate that a special operation is to be performed when the view is checked in. For example, setting the action attribute `sys.delete` requests that an account be deleted.

- **Policy attributes** are values that reflect the indicate how the account has been affected by the Service Provider Account Policy, which is configured on the main Service Provider configuration page of the Administrator Interface. An account lockout occurs in Service Provider when a user has too many consecutive failed login attempts.

- **Runtime attributes** are values that exist only in the view and are never stored on the resource. Some runtime attributes contain information derived from the resource definitions which may assist the application in configuring the user interface. Other runtime attributes may contain state related to UI technologies such as Identity Manager forms. Finally, the application may store arbitrary attributes in the view provided the names do not conflict with other attributes.

  Commonly used runtime attributes include `display`, `info`, and `command`.

Top-Level Attributes

The following table lists the system defined top-level attributes of the view.

Attribute	Description
sys	An object containing internal system attributes related to the composite object. This is functionally similar to the `waveset` attribute in the Identity Manager user view, except that there are fewer items that can be modified. Many of the attributes in this object are action attributes, meaning that they are NOT stored, but setting them causes the check-in to perform certain actions.
info	An object containing metadata about all of the objects linked to the composite object. This is functionally similar to the `accountInfo` attribute in the Identity Manager User view. This object is read-only.
objects	An list of objects containing the attributes of accounts linked to the directory user. This is functionally similar to the `accounts` list in the Identity Manager User view.
display	An object containing Generic Edit Form state. This is set only if you are using Identity Manager XML forms.
command	The command posted to the Generic Edit Form processor. This is set only if you are using Identity Manager XML forms.
policy	Policy-related attributes. See "policy Attributes" on page 42 for more information.

Unlike the Identity Manager User view, the primary attributes of the composite object will be stored as top-level attributes of the view, rather then nested in `waveset` or `accounts[Lighthouse]`. The schema of the composite is variable, though the following attributes will always exist.

Attribute	Description
name	The unique object name. For objects in directories, this is normally the `uid` not the full DN. For accounts, this is the name that the user would use to login.
password	The password if this object represents an account. When the view is fetched, this will not have the current password. It may be set by the application to change the password.
resources	A list of Resource object names representing the assigned resources.
applications	A list of Application object names representing the assigned applications. This is the same as what Identity Manager refers to as "Resource Groups" in the user interface.
roles	A list of Role object names representing the assigned roles.

In addition to the standard attributes, the following attributes will usually be defined in the composite object schema if the object represents a user account.

- firstname
- lastname
- fullname
- email

sys Attributes

The sys attribute retrieves and sets system characteristics. The following table defines the attributes currently defined under the sys attribute. The attribute will be shown as paths from the root of the view.

Action attributes do not have a value when checked out. An asterisk refers to the name of a resource. An attribute such as `sys.links[*].name` would be expanded to a value such as `sys.links[HR Database].name` .

Attribute	Description
sys.identity	The full native identity of the object. For directory objects, this will be the DN. When creating new objects, this will override the identity template defined in the Resource. For existing objects, it must not be changed.
sys.delete	An action attribute that may be set to `true` to indicate that this object should be deleted.
sys.disable	An action attribute that may be set to `true` to disable an account. If it is set to `false`, then the account is enabled. This is an action-only attribute. When the view is checked out, the attribute will not have a value.
sys.newIdentity	An action attribute that may be set to another object identity to indicate that the object should be renamed. Not all resources support changing identities.
sys.resetPassword	An action attribute that may be set to `true` to indicate that the password should be reset.
sys.expirePassword	An action attribute that may be set to `true` to indicate that the password should be expired.
sys.form	The name of the form used to display or process this view, set only when Identity Manager XML forms are being used.
sys.noDefaultForm	An action attribute that may be set to true to prevent any default user form from being used.

Attribute	Description
sys.links	A list of objects holding information about the objects linked to the composite. This is functionally equal to the waveset.accounts in the user view.
sys.lock	If this boolean action is set to `true`, then when the IDMXUser view is checked-in (or refreshed), then the IDMXUserViewer will run the "Lock Account Rule" to set the appropriate values in the view to lock the account. If this rule is not specified in the configuration and sys.lock is `true`, an exception is thrown.
sys.locked	Set to true if the account has been locked explicitly or due to too many failed login attempts. The IDMXUserViewer runs the "Is Account Locked Rule" to determine if the account is locked.
sys.unlock	If this boolean action is set to `true`, then when the IDMXUser view is checked-in (or refreshed), the IDMXUserViewer will run the "Unlock Account Rule" to set the appropriate values in the view to unlock the account. If this rule is not specified in the configuration and sys.lock is `true`, an exception is thrown.
sys.targets	A list of resources, services, or applications that should be targeted on the check-in. The list can include the Resource, Service, or Application PersistentObjects themselves, or more usefully just their names. If no targets are specified, then the provisioner will provision to all assigned resources, service, and applications.
sys.links[*].name	A unique name used to identify the linked object in the view. Usually this is the same as the name of the Resource containing the linked object, but may be qualified if there is more than one linked object from this Resource.
sys.links[*].resource	The name of the Resource containing the linked object. Often the same as sys.links[*].name, but not necessarily.
sys.links[*].type	The type of the Resource containing the linked object.
sys.links[*].identity	The full native identify of the object. For directory objects, this will be the DN.
sys.links[*].created	Set to `true` if the object is believed to exist.
sys.links[*].disabled	Set to `true` if the object is currently disabled.
sys.links[*].locked	Set to `true` if the object is currently locked.
sys.links[*].fetched	Set to `true` if the current attributes of the object have been fetched.
sys.links[*].attributes	An object containing last known attribute values of the linked object. This will be set only when the application requests that the current values be fetched. They are not stored permanently in the composite.

Attribute	Description
sys.txn.waitForFirstAttempt	This attribute dictates how control returns to the caller when an IDMXUser view object is checked in. If set to `true`, the check-in operation will block until the provisioning transaction has completed a single attempt. If set to `false`, the check-in operation will return control to the caller before attempting the provisioning transaction. It is recommended to enable this option. If asynchronous processing is disabled, then the transaction will have either succeeded or failed when control is returned. If asynchronous processing is enabled, then the transaction will continue to be retried in the background.
sys.txn.enableAsynchronous	This attribute controls whether processing of provisioning transactions continues after the check-in call returns. Since only a single attempt will be made synchronously, this option must be enabled if retrying transactions is desired.
sys.txn.asynchronousMS	An upper bound expressed in milliseconds of how long the server will retry a failed provisioning transaction. This setting complements the retry settings on the individual resources, including the master LDAP directory. For example, if this limit is reached before the resource retry limits are reached, the transaction will be aborted. If the value is negative, then the number of retries is only limited by the settings of the individual resources.
sys.txn.persistImmediately	If set to `true`, provisioning transactions will be written to the Transaction Persistent Store before they are attempted. Enabling this option might incur unnecessary overhead since most provisioning transactions will succeed on the first attempt. It is recommended to disable this option unless the waitForFirstAttempt attribute is disabled.
sys.txn.persistOnAsync	If set to `true`, provisioning transactions will be written to the Transaction Persistent Store before they are processed asynchronously. If the waitForFirstAttempt attribute is enabled, then transactions that need to be retried will be persisted before control is returned to the caller. If the waitForFirstAttempt attribute is disabled, then transactions will always be persisted before they are attempted. It is recommended to enable this option.
sys.txn.persistOnEachUpdate	If set to `true`, provisioning transactions will be persisted after each retry attempt.

The sys.txn attributes correspond to fields displayed on the **Default Transaction Execution Options** section of the **Edit Transaction Configuration** page of the Administrator Interface. This page sets the values globally.

The global values can be overridden in a user form as follows.

```
<Field name='sys.txn.persistImmediately'/>
   <Default>
```

```
      <s>true</s>
    </Default>
</Field>
```

objects Attributes

The objects attribute contains a list of objects, each holding the editable attributes of the linked objects. It is similar to the `accounts` attribute in the Identity Manager User view. Like the Identity Manager User view, the attributes in each object are mostly arbitrary and defined by the Resource schema. The following attributes are defined by the system and will always exist.

Attribute	Description
objects[*].name	A unique name used to identify the linked object in the view. Usually this is the same as the name of the Resource containing the linked object, but may be qualified if there is more than one linked object from this Resource. This name may be used to locate an object in the info attribute.
objects[*].password	May be used to set the password for accounts that support passwords. This value will initially be null for new accounts. When editing existing accounts, it will be non-null to indicate that a password has been set, but the value will not be the actual password.
objects[*].locked	Initially set to true if the linked object is an account and has been locked. May be changed to change the lock status of the account. This attribute is valid only if the associated resource supports account locking. To unlock a locked account, you must explicitly set this value to the string `false` . If you leave the value null, the current lock status will not be changed.
objects[*].sys.identity	The full native identity of the object. For directory objects, this will be the DN. When creating new objects, this will override the identity template defined in the Resource. For existing objects, it must not be changed.
objects[*].sys.delete	An action attribute that may be set to `true` to indicate that this object should be deleted.
objects[*].sys.disable	An action attribute that may be set to `true` to disable an account. If it is set to `false`, then the account is enabled. This is an action-only attribute. When the view is checked out, the attribute will not have a value.
objects[*].sys.resetPassword	An action attribute that may be set to true to trigger a password reset. This will result in a randomly generated password being assigned.
objects[*].sys.expirePassword	An action attribute that may be set to `true` to indicate that the password should be expired.
objects[*].sys.unlink	An action attribute that may be set to `true` to indicate that this object should be unlinked.

Attribute	Description
objects[*].sys.newIdentity	An action attribute that may be set to another object identity to indicate that the object should be renamed. Not all resources may support identity changes.

info Attributes

The info attribute contains metadata about the objects liked to this composite. It currently does not have much, but will evolve to contain information similar to that in the accountInfo attribute of the Identity Manager User view. There will however be much more control over the amount of information included.

Attribute	Description
info.resourceTypes	A list of resource type names for all resources assigned to this user.

policy Attributes

Account lockout occurs when a user has too many consecutive failed login attempts. This applies to both password-based login attempts and login attempts based on authentication questions. Separate limits for password-based and question-based account lockout are specified on the "Account Policy" section on the main configuration page. Accounts that are locked out can be explicitly unlocked by an administrator or implicitly when the lock expires, such as after one hour.

Since locking out accounts is LDAP-vendor specific, Service Provider allows you to configure rules that operate on the IDMXUser view to determine if an account is locked out, to update the view to lock an account, and to update the view to unlock an account.

The policy.questions[*] attributes will not be included in the IDMXUser view or updated unless the buildAuthQuestions option is set in the form.

Attribute	Description
policy.failedPasswordLoginAttemptsCount	The number of consecutive password-based failed login attempts. When the user logs in successfully (either with a password or authentication questions), this value is reset. If this value is not present, it is assumed to be zero. (integer: read-only)

Attribute	Description
policy.failedQuestionLoginAttemptsCount	The number of consecutive authentication question-based failed login attempts. When the user logs in successfully (either with a password or authentication questions), this value is reset. If this value is not present, it is assumed to be zero. (integer: read-only)
policy.questions[*].id	The unique identifier that is used to associate this question with one defined in the policy. This is a read-only attribute.
policy.questions[*].question	The question text, which can be displayed to the user. This attribute is read-only.
policy.questions[*].answer	The user's answer for the question, if specified.
policy.questions[*].loginInterface	The login interface with which this policy question is associated. Its value is a unique message catalog key for each loginInterface.

IDMXUser View Differences

The following sections provide a summary of the differences between the IDMXUser view, and views defined in Identity Manager.

Identity Manager User View

Top-Level Attributes

Identity Manager User View	IDMXUser View
password	Not applicable
global	Not applicable
update	Not applicable

waveset Attributes

Identity Manager User View	IDMXUser View
waveset.form	sys.form
waveset.id	sys.id
waveset.accountId	accountId (actual attribute name is name assigned in the schema map)

Identity Manager User View	IDMXUser View
waveset.password	password (actual attribute name is assigned in the schema map)
waveset.email	email (actual name is assigned in the schema map)
waveset.disabled	Not applicable
waveset.roles	roles
waveset.resources	resources
waveset.policies	policy
waveset.applications	applications
waveset.organization	Not applicable
waveset.organizationId	Not applicable
waveset.policies	Not applicable
waveset.adminRoles	Not applicable
waveset.capabilities	Not applicable
waveset.creator	Not applicable
waveset.createDate	Not applicable
waveset.lastModifier	Not applicable
waveset.lastModDate	Not applicable
waveset.backgroundSave	Not applicable
waveset.attributes	Not applicable
waveset.original	Not applicable

waveset.accounts Attributes

Identity Manager User View	IDMXUser View
waveset.accounts	sys.links
waveset.accounts[].resource	sys.links[].resource
waveset.accounts[].id	Not applicable
waveset.accounts[].accountId	sys.links[].identity
waveset.accounts[].accountGUID	sys.links[].guid

Identity Manager User View	IDMXUser View
waveset.accounts[].accountDisplayName	sys.links[].displayName
waveset.accounts[].tempId	Not applicable
waveset.accounts[].created	sys.links[].created
waveset.accounts[].disabled	sys.links[].disabled
waveset.accounts[].attributes	sys.links[].attributes
waveset.accounts[].password	sys.links[].attributes.password
waveset.accounts[].resourceAttributes	Not applicable
waveset.accounts[].properties	Not applicable
waveset.accounts[].templateParameters	Not applicable

accounts Attributes

Identity Manager User View	IDMXUser View
accounts[]	objects[]
accounts[].identity	objects[].sys.identity
accounts[].*UserDefined*	objects[].*UserDefined*
accounts[Lighthouse].firstname	firstname (attribute name assigned in schema map)
accounts[Lighthouse].lastname	lastname (attribute name assigned in schema map)
accounts[Lighthouse].fullname	fullname (attribute name assigned in schema map)
accounts[Lighthouse].*UserDefined*	*UserDefined*

accountInfo Attributes

Identity Manager User View	IDMXUser View
accountInfo	info
accountInfo.typeNames	info.resourceTypes
accountInfo.types	Not applicable
accountInfo.accounts	info.objects
accountInfo.accounts[Lighthouse]	info.master

resourceAccounts View

Identity Manager User View	IDMXUser View
selectAll	Not applicable
resourceAccounts.currentResourceAccounts	objects
resourceAccounts.tobeCreatedResourceAccounts	info.objects
resourceAccounts.tobeDeletedResourceAccounts	info.objects
resourceAccounts.currentResourceAccounts[].attributes	objects[]
resourceAccounts.currentResourceAccounts[].selected	Not applicable

Note – The mappings for `info.objects` are functionally similar to their resource accounts views counterparts, but they are not structurally similar.

In general, in Identity Manager, the resource accounts views operate by setting the `selected` attribute in the `currentResourceAccounts` list to `true`, which will then have different behavior for each view type.

In Service Provider, accounts are not selected with a boolean attribute. Instead, each operation has an `action` attribute that is initially null, and when set, triggers the operation.

All resource accounts views provide a way to update arbitrary account attributes in addition to performing an operation.

Most allow attributes to be placed in:

```
resourceAccounts.currentResourceAccounts[].attributes
```

The Rename view uses:

```
accounts[]
```

In the IDMXUser view, you always place attributes you want to modify in:

```
objects[]
```

password View

The desired password is simply set in the following attributes:

```
password
objects[].password
```

There is no automatic synchronization of passwords from the top level `password` attribute to the `password` attribute on the individual resource accounts. To pre-expire the password, set the following attributes:

```
sys.expirePassword
objects[].sys.expirePassword
```

Disable and Enable Views

In Identity Manager, accounts are enabled or disabled by checking out the Enable or Disable view and setting the value of

`resourceAccounts.currentResourceAccounts[].selected` to `true`.

In the IDMXUser view, you set the following action attributes to `true` or `false`.

```
sys.disable
objects[].sys.disable
```

Rename View

In the Identity Manager Rename view, the new name is specified in the top-level field `newAccountId`, which is then propagated to those resource accounts selected in the `currentResourceAccounts` list. In the IDMXUser view, renames are requested by setting the following attributes to the desired identity, which for LDAP resources must be the full DN.

```
sys.newIdentity
objects[].sys.newIdentity
```

Deprovision View

The Identity Manager deprovision view has a complex structure designed for use with interactive forms. The three operations that may be requested through this view are:

`unassign`. Removes the assignment and deletes the account.

`unlink`. Removes the account link without deleting.

`delete`. Deletes the account, but leave the assignment.

In the IDMXUser view, you perform an unassign simply by removing a name from the `roles`, `resources`, or `applications` lists.

To perform an unlink, you may remove an object from the `sys.links` list, or set the following action attribute to `true`:

```
objects[].sys.unlink
```

To perform a delete without unassigning set the following command attribute to `true`.

```
objects[].sys.delete
```

To delete the directory user, set the following attribute to `true`:

```
sys.delete
```

◆ ◆ ◆ CHAPTER 5

Other Objects in Service Provider

This chapter describes how the following objects differ between Identity Manager and Service Provider:

Service Provider User Forms

All user forms that access account information for Service Provider accounts must use the IDMXUser view to reference attributes on their Service Provider Directory account and linked resource accounts.

If you create your own user form, and the form is to be displayed as an option on the **Edit Main Configuration** page of the Administrator Interface, the `authType` Configuration attribute must be assigned the value `SPEUserForm`. Otherwise, this attribute may be omitted. Service Provider user forms operate on the IDMXUser view instead of the Identity Manager User view. Therefore, a different authType is useful for distinguishing between the forms.

The following example illustrates the use of the `authType` attribute.

```
<Configuration authType='SPEUserForm' wstype='UserForm' name='My Service Provier User Form'>
```

The **Edit Main Configuration** page of the Administrator Interface allows an administrator to select a default End User, Administrator, and Synchronization user form. If no End User or Synchronization user form is defined in the Administrator Interface, then no form will be used, unless a form is specified in the `sys.form` attribute in the IDMXUser view. If a default Administrator user form is not selected, then delegated administrators will not be able to create

or edit accounts from the Administrator Interface. However, if the `sys.noDefaultForm` attribute is set to `true` in the IDMXUser view, the default user form specified in the Administrator Interface will not be used.

End User Form

The end user form is typically a simple form that defines a few fields and possibly provides default values. The `display.session` and `display.speContext` attributes should not be called, because the results are not displayed on the Administrator Interface.

Administrator User Form

The administrator user form determines what is displayed on the Create or Edit Service Provider Users page. This form will typically be more complex than the end user form, as it can be used to perform tasks such as retrieving lists of available resources and determining which attributes can be edited.

Synchronization User Form

The Service Provider Synchronization task uses the synchronization form to process accounts. This task ensures that account changes on one resource are propagated to other resources, including the LDAP master directory.

The Synchronization user form performs the same general functions as an Identity Manager Active Sync user form. It translates an arbitrary set of incoming account attributes into the IDMXUser view. See "Active Sync User Forms" on page 53 for more information.

Account and Password Policies

Service Provider account and password policies behave differently than those for Identity Manager:

- There can be only one account policy defined for the entire Service Provider user population. This policy can be set on the main Service Provider configuration page.
- The Service Provider account policy does not support the following options that are available for Identity Manager users
  - **Expires in**
  - **Warning time before expiration**
  - **Reset Option**

- **Reset temporary password expires in**
- **Reset Notification Option**
- **Passwords may be changed or reset**
- **Enforce Answer Policy at Login**
- **Allow User Supplied Questions**

  These options are not displayed when editing the Service Provider Policy object.

The Service Provider account policy can refer to an accountId and a password policy. Service Provider does not maintain the password history. Password policies, however, allow setting the **Number of Previous Passwords that Cannot be Reused** and **Maximum Number of Similar Characters from Previous Passwords that Cannot be Reused** options. These options are ignored by Service Provider.

- Account lockout in Service Provider occurs when a user has too many consecutive failed login attempts. This applies to both password-based login attempts and login attempts based on authentication questions.
- The sample account policy Service Provider Policy sets separate limits for password-based and question-based account lockouts. Accounts that are locked out can be explicitly unlocked by an administrator or implicitly when the lock expires (such as after one hour).

Rules

Because the mechanism for locking out accounts varies for each LDAP vendor, Service Provider allows you to configure rules that operate on the IDMXUser view to determine if an account is locked out, to update the view to lock an account, and to update the view to unlock an account. These rules are selected on the **Edit Main Configuration** page.

Is Account Locked Rule

The rule selected as the "Is Account Locked Rule" determines if an account is locked. The full IDMXUser view is available to this rule. The rule takes the following arguments:

- `maxFailedPasswordLogins` — An integer declaring the maximum number of failed password logins before an account is locked according to the Service Provider System Account Policy.
- `maxFailedQuestionLogins` — An integer declaring the maximum number of failed question logins before an account is locked according to the Service Provider System Account Policy.

The rule should return true only if the account is locked.

The sample rule "Service Provider Example Is Account Locked Rule" operates on Sun Java System Directory Server 5.x. This rule expects that the accountUnlockTime and passwordRetryCount account attributes are defined in the LDAP resource schema map.

Lock Account Rule

The rule selected as the "Lock Account Rule" sets attributes in the IDMXUser view that cause an account to be locked. The full IDMXUser view is available to this rule. In addition, it takes the following argument:

`lockExpirationDate`: A possibly null `java.util.Date` at which the lock should expire.

This rule should update the IDMXUser view so that the LDAP account will be locked when the view is checked in.

The sample rule "Service Provider Example Lock Account Rule" on Sun Java System Directory Server 5.x. This sample rule expects that the accountUnlockTime and passwordRetryCount account attributes are defined in the LDAP resource schema map.

Unlock Account Rule

The rule selected as the "Unlock Account Rule" on the main configuration page sets attributes in the IDMXUser view that cause an account to be unlocked. The full IDMXUser view is available to this rule. The rule takes no additional arguments.

This rule should update the IDMXUser view so that the LDAP account will be unlocked when the view is checked in.

The sample rule "Service Provider Example Unlock Account Rule" operates on Sun Java System Directory Server 5.x. This rule expects that the accountUnlockTime and passwordRetryCount account attributes are defined in the LDAP resource schema map.

Resources

Note the following items when creating and configuring resources for use with Service Provider:

- You must configure the user directory as a resource. The directory can be maintained from the Service Provider Configuration pages in the Administrator Interface.
- Any resources that are to be synchronized with the user directory should be configured from the Edit Synchronization Policy resource action, and configured for the target object type "Service Provider User."

- Service Provider stores all of its meta information in the LDAP attribute mapped to `xml` in the resource definition schema map. (For example, `jpegPhoto`) The size of the data stored in this attribute is usually small (<1KB) but can be larger, depending on the number of resources assigned to the user.

  If the Retro Changelog is enabled in your Sun Java System Directory Server, add the attribute that corresponds to the `xml` LDAP attribute (for example, `jpegPhoto`) to the list of `ignore_attributes` in the Retro Changelog plugin's configuration. This can be accomplished by using the Directory Server administration console.

The default user form and rules associated with each resource might need to be customized to work with Service Provider. See the following sections for more details:

- "Active Sync User Forms" on page 53
- "Rules" on page 53

Active Sync User Forms

The following list describes the differences between user forms in Identity Manager and Service Provider:

- In Identity Manager, both the Synchronization Input form and the default user form are processed. In Service Provider, only the Synchronization Input form is processed.
- The IDMXUser view is used instead of a User view. See Chapter 4, "IDMXUser View," for more information.
- In both Identity Manager and Service Provider, all of the attributes received are placed under the `activeSync` top-level attribute.
- Identity Manager also sets the following attributes to contain information about the resource definition itself:

  ```
  activeSync.resourceName
  activeSync.resourceId
  activeSync.resource
  ```

  Other attributes are culled from the resource. Service Provider does not set these.

Rules

The nature of the following rules varies in Service Provider:

- Correlation rules
- Confirmation rules
- Delete Rules

Correlation Rule

Because reconciliation is not performed on a Service Provider resource, a correlation rule is invoked on-demand to discover existing account information.

In the Identity Manager User view, the `account` attribute may contain numerous account attributes. In the IDMXUser view, these account attributes are defined at the top-level of the view. For example, in Identity Manager, a correlation rule might make a reference to `<ref>account.accountId</ref>`. In Service Provider, the reference would simply be `<ref>accountId</ref>`.

Confirmation Rules

A Identity Manager confirmation rule may reference the following attributes:

`account` — The attributes from the Active Sync account

`user` — The User view

A Service Provider confirmation rule sets the same attributes, but the contents of the `user` attribute contains only the user attributes stored in the directory. It will not contain a full IDMXUser view.

Delete Rule

An Identity Manager delete rule may reference the following attributes:

`activeSync` — The attributes from the Active Sync account

`account` — Identical to `activeSync`.

In a Service Provider delete rule, the Active Sync account attributes are at top-level. They are not wrapped in either `activeSync` or `account`.

Link Correlation Rule

The `linkTargets` IDMXUser view option allows the caller to specify the list of resources that should be targeted for linking. When using forms, the list can be provided as a form property with the same name. Form properties are assimilated into view options when the IDMXUser view is checked in.

A link correlation rule selects resource accounts that the user might own. Given the view of the user, a link correlation rule returns an identity, a list of identities, or an option map.

If the rule returns an option map, then the view handler uses the map to look for resource accounts and obtains a list of identities that satisfy these options. For example, the `searchFilter` option of the `getResourceObjects` FormUtil method can be used to pass a search filter to an LDAP resource adapter.

A link correlation rule must have the `authType` attribute set to `SPERule` with the `subtype` set to `SUBTYPE_SPE_LINK_CORRELATION_RULE`.

Link Confirmation Rule

A link confirmation rule eliminates any resource accounts from the list of potential accounts that the link correlation rule selects. Given the view of the user and the list of candidate resource accounts, a link confirmation rule selects at most one resource account from the candidate list. The view of the user is visible under the 'view' path, while the list of candidates is available under the 'candidates' path.

If the link correlation rule selects no more than one resource account, the link confirmation rule is optional.

Note – Unlike Identity Manager confirmation rules , a link confirmation rule is invoked only once during the linking process.

A link confirmation rule must have the `authType` attribute set to `SPERule` with the `subtype` set to `SUBTYPE_SPE_LINK_CONFIRMATION_RULE`.

Workflow Callouts

Service Provider does not support workflows. However, workflow callouts can be implemented.

Workflow callouts enable the execution of custom code before and/or after a Service Provider transaction is processed. A pre-operation callout is executed before, while the post-operation callout is executed after a transaction is processed.

Callouts are registered with the Service Provider main configuration object in the `workflowCallout.callouts` attribute. The key in the `workflowCallout.callouts` map must be the name of the registered callout, while the value is the class implementing the callout. Registered callouts can be selected on the Service Provider configuration page of the Administrator interface for invocation in both the pre- and post-operation stages and for each transaction type.

Use the IDMXUserViewer to override the system-wide callout settings. By setting the `preOpCallout` or `postOpCallout` view options to the name of the callout, the system-wide callout settings can be overridden.

Callouts return any of the following values: success, failure or pending. The pending result is returned if the callout requires more time to process the transaction, such as a manual interaction. Callouts returning a pending result can resume transactions later. The following table summarizes how Service Provider proceeds in each of the possible scenarios.

	Success	Failure	Pending
Pre-operation callout	The transaction processing resumes.	The transaction is aborted.	The transaction is in a waiting for a response.
Post-operation callout	The transaction completes with success state and may be removed from the store.	The transaction completes with failure state and may be removed from the store.	If "wait for post-operation callout" configuration option is enabled, then transaction remains parked. Otherwise, the transaction completes and may be removed from the store

Asynchronous transaction processing must be enabled if any callout may return a pending result. Otherwise, the transaction will always fail.

Callouts must implement the WorkflowCallout interface. Service Provider wraps relevant information about the transaction into a WorkflowAction object. Callouts return an ActionResult object with success, failure or pending enumerated values.

Callouts invoke the `resumeWorkflowCallback` method in order to resume a transaction that have been parked after returning a pending result. The `com.sun.idm.idmx.txn.CalloutManager` implements this interface.

Sample code that resumes a transaction:

```
WorkflowCallback callback = CalloutManager.getInstance();
   callback.resume(transactionId, ActionResult.SUCCESS);
```

The above mentioned interfaces are in the `com.sun.idm.idmx.api` package. Javadocs are provided in the Service Provider reference kit. The interfaces are still evolving, and future changes are possible.

Delegated Administration

By default, Service Provider implements the organization-based authorization model. In order to grant more fine grain capabilities and scope of control on Service Provider users, the admin role-based authorization model can be enabled using Service Prover user Admin Roles.

For more information about enabling admin Role-based delegated administration, see Business Administrator's Guide.

A User Admin Role may be configured to evaluate several rules to determine the scope of control, capabilities, and dynamic assignment to a user:

- Scope of Control rules
  - "User Search Context Rule" on page 57
  - "User Search Filter Rule" on page 57
  - "After Search Filter Rule" on page 58

  Capabilities rule
  - "Capabilities Per User Rule" on page 58

  Assign to Users rules
  - "Assign To User Rule" on page 59
  - "Assign To Service Provider User Rule" on page 59

    Sample rules are located in the `$WSHOME\sample\rules\adminRoleRules.xml` file.

User Search Context Rule

A User Search Context rule is evaluated when searching for Service Provider users. It returns a valid LDAP distinguished name (DN). This DN serves as the base context for searching users. The authType for the rule must be set to `SPEUsersSearchContextRule` .

The rule is passed the following arguments:

`context` — Specifies current user's Identity context (session).

`runAsUser` — The User view of the user the rule will run as. This is a null argument if `runAsIDMXUser` is specified.

`runAsIDMXUser`— The IDMXUser view of the user the rule will run as. This is a null argument if `runAsUser` is specified.

User Search Filter Rule

The User Search Filter rule is evaluated when searching for Service Provider users. It returns an LDAP-compliant filter string that will be logically ANDed with the user's search filter when listing Service Provider users. The authType for the rule must be set to `SPEUsersSearchFilterRule`.

The rule is passed the following arguments:

`context` — Specifies current user's Identity context (session).

`runAsUser` — The User view of the user the rule will run as. This is a null argument if `runAsIDMXUser` is specified.

`runAsIDMXUser` — The IDMXUser view of the user the rule will run as. This is a null argument if `runAsUser` is specified.

After Search Filter Rule

The After User Search Filter rule is evaluated when searching for Service Provider users. It runs after the initial search is performed against the Service Provider user directory. It returns a list of objectIds the requesting user is allowed to list and view.

This type of rule can be used to determine whether a user should be in the requesting user's scope of control. To accomplish this, the rule can reference non-LDAP user attributes, such as LDAP group membership. The rule can also be used when the filter decision needs to be made using a repository other than the Service Provider user directory, such as an Oracle database or RACF.

The authType of this rule must be `SPEUsersAfterSearchFilterRule`.

The rule is passed the following arguments:

`context` — Specifies current user's Identity context (session).

`runAsUser` — The User view of the user the rule will run as. This is a null argument if `runAsIDMXUser` is specified.

`runAsIDMXUser` — The IDMXUser view of the user the rule will run as. This is a null argument if `runAsUser` is specified.

`objectType` — Specifies the type of object, such as IDMXUser, that the rule filters.

`objectIds` — Specifies a list of objects the rule filters.

`conditions` — Specifies a list of AttributeConditions.

Capabilities Per User Rule

In the context of a Service Provider user Admin Role, capabilities specify which capabilities and rights the requesting user has on the Service Provider user for which access is being requested. The Capabilities Per User rule is evaluated when a request is made to view, create, modify, or delete an Service Provider user. It must be specified as a rule with authType `CapabilitiesOnSPEUserRule`.

The list of capabilities returned from the rule can include both existing Identity Manager capability names (such as "Service Provider Create User") and Identity Manager right names (such as "Modify" and "Execute").

The rule is passed the following arguments:

`context` — Specifies current user's Identity context (session).

`runAsUser` — The User view of the user the rule will run as. This is a null argument if `runAsIDMXUser` is specified.

`runAsIDMXUser` — The IDMXUser view of the user the rule will run as. This is a null argument if `runAsUser` is specified.

`object` — Specifies the name of an object, if an object exists. Otherwise, null.

`objectType` — Specifies the type of object, such as IDMXUser, that the rule filters.

`object.identity` — Specifies the DN of the Service Provider user on which the request is being made.

`object.attributes` — Defines a map of attribute name/value pairs of the Service Provider user on which the request is being made. Sample name/value pairs include `sn=Smith` and `cn=gsmith` .

Assign To User Rule

Service Provider user Admin Roles can be dynamically assigned to Identity Manager users by specifying an Assign to User rule. This rule is evaluated at login time to determine whether to assign the authenticating Identity Manager user the AdminRole. The authType of this rule must be `UserIsAssignedAdminRoleRule` .

The rule is passed the following arguments:

`context` — Specifies current user's Identity context (session).

`runAsUser` — Retrieves the User view of requesting user.

The output is a string or boolean `true` if the authenticating user should be assigned this AdminRole. Otherwise, `false` or null should be returned.

Assign To Service Provider User Rule

Service Provider User AdminRoles can be dynamically assigned to Service Provider users by specifying an Assign to Service Provider User rule. This rule is evaluated at login time to determine whether to assign the AdminRole to the authenticating Service Provider user. The authType of the rule must be `SPEUserIsAssignedAdminRoleRule`.

The rule is passed the following arguments:

`context` — Specifies current user's Identity context (session).

`runAsIDMXUser` — Retrieves the IDMXUser view of requesting user.

The output is a string or boolean `true` if the authenticating user should be assigned this AdminRole. Otherwise, `false` or null should be returned.

◆ ◆ ◆ CHAPTER 6

Service Provider User Interface

The bundled sample end-user pages demonstrate the features of Service Provider by providing examples for registration and self-service typical in extranet service provider environments. Since the samples have been developed with extensibility and customization in mind, typical tasks such as changing the look and feel, modifying navigation rules between pages, or displaying locale-specific messages are straightforward to perform.

In addition to auditing self-service and registration events, notification to the affected user can be sent using e-mail templates. Examples of using account ID and password policies as well as account lockout are also provided. Developers can also implement Identity Manager forms. The modular authentication service implemented as a servlet filter can be extended or replaced if necessary. This allows integration with access management systems like Sun Java System Access Manager.

Initial Configuration for the Sample User Pages

To demonstrate the features of the sample user pages, you need to execute the following configuration steps after the initial setup and configuration of Identity Manager has been completed:

1. Setup the sample Service Provider resource.
2. Setup Service Provider authentication policy.
3. Configure mail notification settings.
4. Review the sample users.

Note – Perform all initial configuration from the Identity Manager Administrator Interface. Refer to Business Administrator's Guide for detailed information about configuring this product.

Sample Service Provider Resource

The Identity Manager installation process creates a sample LDAP resource named Service Provider End-User Directory. The schema map defines all the account attributes required for the user pages.

However, you must update the Resource Parameters to specify the host name, user DN, password, and other parameters required to connect to an LDAP directory server. This resource can be used as both configuration and end-user resource for Service Provider.

The resource is defined in the `$WSHOME/sample/speEndUserResource.xml` file.

Sample Service Provider Policy

The installation process also creates a policy named Service Provider Policy This policy uses the default Identity Manager account and password policies but provides the following customized secondary authentication policy options:

- The policy is valid for the Service Provider End User Interface only.
- The maximum number of failed login attempts is 3.
- The default authentication questions are "What is your favorite color?" and "What is your favorite movie?"
- To be authenticated on the Forgot Username or Forgot Password pages, the user must answer all the questions correctly.

The Service Provider Policy is defined in the `$WSHOME/sample/spePolicy.xml` file.

Mail Notification Settings

By default, most of the sample pages are configured to send an email upon completion of the task demonstrated on the page. If you want to turn notification off, then edit the notification section of the SPEUserPages configuration object. The configuration allows to enable/disable email notification for every page individually. See "Configuring the Filter" on page 73 for more details.

Make sure that you configure the SMTP server. To do this, click **Configure**, then **Servers** and navigate to the Email Template tab for the SMTP server.

Sample Users

The SPE Sample Users configuration object provides several sample users. These users are required for the registration pages to function. The sample pages are set up so that a user registering must verify his or her relationship with the provider.

The sample registration pages work out of the box with the provided sample users. The validate page requires the email, firstName and lastName attributes to match before the user can advance to the validation page. Use the debug pages to view the attributes of these sample users.

Sample User Pages Overview

This section describes each sample user page provided with Service Provider. The features, such as page processor class and view handler, are discussed in Chapter 7, "Implementing Custom User Pages."

Login Page

The login page is the entry into the Service Provider User Interface. The password is validated against the password in the LDAP directory. An error is displayed if the user cannot be found in the directory or if the password is invalid.

Authentication occurs through the `com.sun.idm.idmx.web.AuthFilter` servlet filter. To change the filter or its initialization parameters, modify the `$WSHOME/WEB-INF/web.xml` file. See "Configuring the Filter" on page 73 for more details.

If the Service Provider Account Policy has been configured to lock out an account when the user does not login successfully after a specified number of attempts, the user is redirected to another page that states the account is locked. In addition, Service Provider sends the user an email that states the account is locked.

The following table summarizes the structure of this page.

Feature	Name
Page Processor Class	LoginForm
View Handler	IDMXNoopViewer
Forms	Service Provider End-User Login
Email template	Service Provider End-User Profile Locked
Configuration object attributes	Not applicable
Audit event	Not applicable

Registration Page

The Login page provides a link to a registration page that allows prospective users to enroll for the service. The default registration pages implement the following logic

- The user is prompted to provide application or business-specific information to validate relationship with the service provider. This information does not include enrollment information required by Service Provider. In the sample user pages, the user is prompted for the first name, last name and email address fields to verify this relationship. The SPE Sample Users configuration object lists all the values that are accepted on this page.

  This validation step can be skipped if the `enrollment.validation.enabled` configuration setting is set to `false`.
- The user provides the required information and user is validated to have an existing business relationship.
- The user is prompted to provide the required profile enrollment information, including the username, password, and home phone number. Fields shared by the validation page and the enrollment page are automatically filled out. If the Service Provider Account policy is also configured, then the user must also complete the authentication challenge questions. If the `enrollment.privacypolicy` option is enabled, then the privacy policy is also displayed and must be accepted before completing the registration. (The text of the privacy policy is defined in the `IDMXMessages.properties` file.)
- The system displays a message indicating that a new profile has been created, and that an email has been sent to the provided notification address.
- The user is then redirected to login page.

The following table summarizes the structure of this page.

Feature	Name
Page Processor Class	EnrollmentForm
View Handler	IDMXUser
Forms	■ Service Provider Enrollment Main Form ■ Service Provider Enrollment Validation Form ■ Service Provider Enrollment Form ■ Service Provider End-User Forms Library
Email template	Service Provider End-User Registration Template
Configuration object attributes	■ enrollment.validation.enabled ■ enrollment.privacypolicy ■ notification.registration
Audit event	Create

Forgot Username Page

A user accesses the Forgot Username page to retrieve his or her login ID. The user must supply the telephone number stored in the `telephoneNumber` attribute on the directory and a valid email address. The login ID will be sent to the specified email address.

The following table summarizes the structure of this page.

Feature	Name
Page Processor Class	ForgotUsernameForm
View Handler	IDMXLookupUsernameViewer
Forms	Service Provider End-User Forgot Username
Email template	Service Provider End-User Username Recovery
Configuration object attributes	■ lookup-attributes.name ■ lookup-attributes.title ■ lookup-attributes.required ■ notification.recovery
Audit event	usernameRecovery

Failure occurs if an account cannot be found with the specified telephone number and email address, or if multiple accounts are found with the given information.

The form can also use the “auditEventType” form property to instruct the viewer about which type of audit event to log.

Forgot Password Page

A user selects the **Forgot password?** link on the login page to display a page similar to the Forgot Username page. The user must first supply the telephone number stored in the `telephoneNumber` attribute on the directory and a valid email address. Next, the user is prompted to provide answers to authentication questions.

If the user has not previously answered their authentication questions or if authentication questions are not configured, an error is displayed. If the correct answers are given to the authentication questions, either a password is generated and emailed to the user, or the user is redirected to a page allowing them to reset their password. The `password` attribute in the SPEUserPages configuration object determines which action the system takes.

If configured in the Service Provider Account Policy, the account can be locked after a specified number of failed attempts to answer challenge questions.

The following table summarizes the structure of this page.

Feature	Name
Page Processor Classes	■ ForgotPasswordForm ■ UserQuestionForm
View Handlers	■ IDMXLookupUsernameViewer ■ IDMXUserQuestionViewer
Forms	■ Service Provider End-User Forgot Password ■ Service Provider End-User Reset Password
Email template	Service Provider End-User Reset Password
Configuration object attributes	■ lookup-attributes.name ■ lookup-attributes.title ■ lookup-attributes.required ■ notification.passwordreset ■ password.reset-mode
Audit event	challengeResponse (for success and failure)

The form can also use the "auditEventType" form property to instruct the viewer about which type of audit event to log.

Change Password Page

Clicking the My Profile tab in the navigation bar takes the user to a form that allows the user to change his LDAP directory password. The user is prompted to enter his current password, the value of the new password, and a confirmation of the new password.

If the current password is valid, the new password matches its confirmation and also passes the password policy defined for the LDAP resource, then the user's password is modified to the new value. A notification email message is sent to the user's notification address, and an audit event indicating that the user has been updated is generated.

If any of the validations fail, error messages are displayed so the user can correct the form entry and resubmit.

The following table summarizes the structure of this page.

Feature	Name
Page Processor Class	ChangePasswordForm
View Handler	IDMXUser
Forms	Service Provider End-User Change Password
Email template	Service Provider End-User Change Password
Configuration object attributes	notification.passwordchange
Audit event	Update

Change Username Page

This page allows the user to change his or her user name in Service Provider. The provided form makes the following checks on the new user name:

- Checks whether the new user name is already in use.
- If an account ID policy is in use, checks that the user name meets the policy requirements.

The following table summarizes the structure of this page.

Feature	Name
Page Processor Class	ChangeUserIdForm
View Handler	IDMXUser
Forms	Service Provider End-User Change UserId
Email template	Service Provider End-User Change User Id
Configuration object attributes	notification.useridchange
Audit event	Update

Change Notifications Page

A user's notifications address is the email address defined in the LDAP directory. The form associated with this action allows the user to change the email address where she receives notifications.

The only way to truly validate an email address is to try to send a message to it and verify that it was correctly received. This is usually impractical in a form, so the best we can do is usually to verify that the suggested address has a valid format. In this case, the address is valid if it contains an "@" character. If the new address is not valid, an error is displayed in the form allowing the user to correct the address and resubmit.

If the new address is valid, the user's email address is changed and an update user audit event is generated. In addition, an email message is sent to the old address indicating it will no longer be used for notifications and another message is sent to the new address indicating it will be used for future notification messages.

The following table summarizes the structure of this page.

Feature	Name
Page Processor Class	ChangeUserIdForm
View Handler	IDMXUser
Forms	Service Provider End-User Change Notifications
Email templates	■ Service Provider End-User Change Notifications ■ Service Provider End-User Change Notifications Old Address
Configuration object attributes	notification.emailchange
Audit event	Update

Change Challenge Question Answers Page

The Change Challenge Question Answers page allows the user to edit the answers to challenge questions that were specified during enrollment.

An error message is returned if the answers do not meet requirements of the Answer Quality policy.

The following table summarizes the structure of this page.

Feature	Name
Page Processor Class	ChangeNotificationsForm
View Handler	IDMXUser
Form	Service Provider End-User Change Notifications
Email template	Service Provider End-User Update Authentication Answers
Configuration object attributes	notification.questionchange
Audit event	updateAuthenticationAnswers

Logout Page

Clicking the **Logout** button in the masthead sends the user to an action called `/spe/user/LogoutSubmit.do`. The class associated with this action is `com.sun.idm.idmx.web.LogoutAction` . This class invalidates the user's HttpSession. The "success" forward defined for this action takes the user to the login page.

◆ ◆ ◆ CHAPTER 7

Implementing Custom User Pages

The sample user pages follow the Model-View-Controller design pattern. The Apache Struts Action framework provides navigation between the pages, while the Struts/Tiles templating framework is used to create common look and feel for the application. For information about Struts, see http://struts.apache.org/.

The following diagram shows how a typical request is handled in the sample end-user pages.

How a Typical Request is Handled in Sample End-user Pages

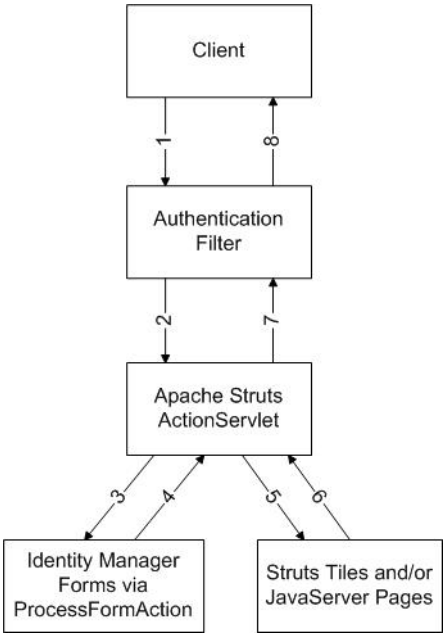

FIGURE 7–1 Flow of a request made within the sample end-user pages.

1. The client requests a page. The servlet container invokes the authentication filter. The filter handles authentication and ensures that protected pages can only be accessed after successful authentication. If a client attempts to access a protected page without proper authentication, the filter redirects the client to the login page.
2. The Struts ActionServlet receives the request and acts according to the directions in the Struts configuration file.
3. For pages backed by Identity Manager forms, the Struts ActionServlet invokes the ProcessFormAction. The ProcessFormAction class is a Struts Action implementation and serves as an adapter to the forms engine. The action specifies the Identity Manager form class (page processor), the view handler, and the custom form.
4. The Struts ActionServlet may redirect the client in case the ProcessFormAction returns an ActionForward other than "success". When a page first displayed, the "success" forward is followed.
5. The HTML content is generated with the help of the Tiles templating framework.
6. The generated HTML is passed back to the ActionServlet.
7. The authentication filter receives the content, but it does not alter it in any way.
8. The generated HTML page is returned to the client.

Authentication and Authorization

The Service Provider Authentication/Authorization servlet filter serves as the default authentication system for the sample user pages. This filter ensures that protected pages can only be accessed after successful authentication. If a client attempts to access to a protected page without proper authentication, the filter would redirect the client to the login page.

Authentication is executed when the user clicks the login button on the login page. The filter attempts to authenticate the user by authenticating the user against the Service Provider directory with the provided credentials. During this process, the system verifies credentials and also determines whether the user's profile has been locked due to multiple login attempt failures. If the profile is locked, then access is denied to the self-service pages and the appropriate error message is displayed.

Pages that require authentication can be accessed using a different path than pages that can be accessed anonymously. This protected path is a configuration option of the filter.

Because pages that require authentication are accessible using a different path than anonymous pages, protecting these pages are easy to configure in any access management system. The sample user pages do not depend on a particular type of authentication nor do they require the password anywhere but in the filter. The sample pages only require the user name for auditing purposes.

When authentication successfully completes, the filter puts the Constants.VAR_SUBJECT variable into the session with the user name as the value. To access pages that require authentication, the subject must be present in the session. When the user logs out of the self-service pages, the session is invalidated. The session timeout configured for the application defines how long the user can be inactive without being "logged out".

Configuring the Filter

The `$WSHOME/WEB-INF/web.xml` file contains the definition the Service Provider Authentication/Authorization filter. This filter handles authentication and ensures that protected pages can only be accessed after successful authentication. If a client attempts to access to a protected page without proper authentication, the filter would redirect the client to the login page.

The Service Provider Authentication/Authorization filter defines the following parameters. Note that the parameters that specify a directory or page must include the path to the web application.

Name	Default Value	Description
protected-pages-path	/spe/user/protected	The full path for pages that require authentication. Pages that do not require authentication should not be placed in this directory.
login-page	/spe/user/Login.do	The full path to the login page where the user is redirected when attempting to access a protected page without proper authentication.
profile-locked-page	/spe/user/ProfileLocked.do	The full path to the page that is displayed when a user attempts to login while the account is locked.
profile-has-been-locked-page	/spe/user/ProfileHasBeenLocked.do	The full path to the page that is displayed when a user exceeds the maximum number of failed login attempts.
preserve-query-string	true	Indicates whether to preserve the query string when a user is redirected to the login page. The allowed values are true or false.

See "Account and Password Policies" on page 50 for information about implementing lockout policies in Service Provider.

Specifying Protected and Public Pages

The `protected-pages-path` parameter in the `web.xml` file specifies the directory in which pages that need authentication must reside. Public pages must reside in a different location.

The following pages are provided in the `$WSHOME/spe/user/protected` directory by default.

- `Form.jsp`— Displays Identity Manager form output.
- `Home.jsp`— The content that appears on the page that is displayed after the user is successfully authenticated.
- `OperationResult.jsp`— Displays a successful operation message in an authenticated context, such as a successful password change.

Other customized files may be added to this directory.

See "Specifying an Action Path" on page 76 for more information about implementing authentication with the default authentication filter.

Integrating with Access Manager

The AuthFilter can also be configured to work in an environment with an access management solution, such as the Sun Java System Access Manager. In these environments, Service Provider is not responsible for logging in the user because the access management solution is used to protect the end user pages.

The normal login process places the user name into the HTTP session. But when the normal process is not used, Service Provider requires the access management solution to put the username in a configured HTTP header before forwarding the HTTP request.

Two configuration attributes in the SPEUserPages configuration object control how the Service Provider end user pages work in this environment.

- `sso-assume-authenticated`— If set to true, the AuthFilter will not redirect to a login page. However, for auditing purposes, the filter requires a user name to associate with each request. Normally, this username is stored on the HTTP session by the login pages. However, since the login pages are not used in an SSO environment, the username is pulled from an HTTP header attribute.
- `sso-user-name-http-header-attr`— Specifies the name of the HTTP header attribute to use.

See "SPEUserPages Configuration Object" on page 80 for more information.

For Sun Java System Access Manager, to ensure that the header is set, edit the `AMAgent.properties` file to include entries similar to the following:

```
com.sun.identity.agents.config.profile.attribute.fetch.mode = HTTP_HEADER
com.sun.identity.agents.config.profile.attribute.mapping[uid] = HEADER_speuid
```

Assuming users login with the `uid` attribute, the Access Manager Policy Agent will store the `uid` value in the `HEADER_idmuid` HTTP header before forwarding requests to the Service Provider User Interface pages.

Integrating with Other Access Management Systems

You can use a third-party access management access system to perform the authentication/authorization instead of the provided filter and still leverage the self-service features of the sample pages. The integration is the easiest if the access management system can provide the name of the authenticated user in the HTTP request header. In this case, steps similar to the Integration with Access Manager can be followed.

If providing the name of the authenticated user in the HTTP request header is not possible, then the following guidelines should be followed to enhance or replace the provided filter:

- All self-service pages that require authentication are under the `/spe/user/protected` folder, all other pages are in the `/spe/user` folder.
- On successful authentication, the Constants.VAR_SUBJECT session attribute must be set to the corresponding user name in Service Provider.

Struts Configuration File

Apache Struts drives the navigation between the sample User Interface pages. The Struts configuration file (`$WSHOME/WEB-INF/spe/config/struts-config.xml`) defines all the navigation rules in the form of Struts `action` definitions. Action definitions typically contain `path`, `type`, and `parameter` attributes. Many definitions also contain one or more `forward` elements that indicate which page should be displayed next.

Specifying an Action Path

Pages that require authentication must specify the protected path in the `path` attribute of the appropriate `action` definition. For example, the following `action` definition for the ChangePassword page includes the same path specified in the `protected-pages-path` parameter in the `web.xml` file.

```
<action path="/spe/user/protected/ChangePassword" type="com.sun.idm.idmx.web.SelfServiceProcessFormAction" ...>
```

The login page does not require the user to be authenticated, so the path of the corresponding action does not include the `protected` directory:

```
<action path="/spe/user/Login" type="com.sun.idm.idmx.web.ProcessFormAction" ...>
```

Specifying an Action Type

The type parameter defines the class that performs the action. Some of the default actions refer to a Struts-defined subclass of the `org.apache.struts.action.Action` class. The following subclasses are provided with the sample user pages.

Class Name	Description
ProcessFormAction	Allows access to the Identity Manager forms engine. This action can take a view handler class, a form name, and a page processor class (in that order) as parameters. See "Specifying an Action Type" on page 76 for more information.

Class Name	Description
SelfServiceProcessFormAction	A subclass of ProcessFormAction that is used for self-service operations, such as changing passwords or user IDs.
RelayAction	Redirects the client to a page after the user has acknowledged a successful operation result, such as a successful password change. Since all confirmation pages use a single content tile (`OperationResult.jsp`), an action forward name is set in a hidden field when the content tile is generated. See "OperationResult Relay Definition" on page 90 for more information.
LogoutAction	Invalidates the current user session.

Specifying Action Parameters

The `parameter` attribute defines arguments that are passed to the class specified in the `type` attribute. If the `type` attribute is `ProcessFormAction` or `SelfServiceProcessFormAction` class, then the value may contain the following options, none of which are mandatory. The options must be specified in the order listed and be separated by commas.

1. The viewer class, such as `IDMXUser` or `IDMXLookupUsername`.
2. The form name or ID.
3. The page processor class.

   Each of these options are described below.

   If the action type is a class other than `ProcessFormAction` or `SelfServiceProcessFormAction`, then the value specified in the `parameter` attribute must match the requirements of the class.
4. The string `secure`.

   This parameter is optional. If it is specified, the server always processes page requests using HTTPS.

Viewer Classes

The following viewer classes may be specified if the `type` attribute is `ProcessFormAction` or `SelfServiceProcessFormAction` class:

- `IDMXUser`— The primary Service Provider view. See "IDMXUser View Reference" on page 36 for detailed information about this class.
- `IDMXLookupUsernameViewer`— Returns a user name that matches the attributes specified in the `lookup-attributes` attribute in the SPEUserPages configuration object. See "SPEUserPages Configuration Object" on page 80 for more information about the lookup-attributes attribute.
- `IDMXUserQuestionViewer`— Manipulates the authentication questions.

- `IDMXNoop`— An empty view handler that provides ability to render a form without any required view schema. This requires a form name passed in through the ViewConstants.OP_FORM option.

Forms

The form argument may contain a form defined in Identity Manager. If no form is specified, viewer class uses the default form.

Page Processors

The default page processor is `com.waveset.ui.util.GenericEditForm` class. The `IDMXUserForm` class, a subclass of `GenericEditForm` , is a custom page processor that allows sending email notifications. Email notifications can be enabled or disabled for each in the sample end user pages. In turn, the `IDMXUserForm` has the following subclasses:

Subclass	Description
ChangeAuthenticationAnswersForm	If email notification is enabled and the authentication answers have been successfully updated, then an email is sent to the user using the "Service Provider End-User Authentication Answers Updated" template. The class also records an appropriate audit event.
ChangeNotificationsForm	If email notification is enabled and the notification address has been successfully updated, then emails are sent to both the old and the new address using "Service Provider End-User Change Notifications" and "Service Provider End-User Change Notifications Old Address" template respectively. The class also records an appropriate audit event.
ChangePasswordForm	If email notification is enabled and the password has been successfully changed, then an email is sent to the user using the "Service Provider End-User Change Password" template. The class also records an appropriate audit event.
ChangeUserIdForm	If email notification is enabled and the user name has been successfully changed, then an email is sent to the user using the "Service Provider End-User Change User Id" template. The class also records an appropriate audit event.
EnrollmentForm	If email notification is enabled and registration has been successfully completed, then an email is sent to the user using the template "Service Provider End-User Registration Template".
ResetPasswordForm	If the `password.reset-mode` attribute in the SPEUserPages object is set to `self`, then when the user answers all the required authentication questions, the password is reset with this class.

The following table lists the additional subclasses of GenericEditForm.

Subclass	Description
UserQuestionForm	This page processor class is used when the user forgets his password and attempts to login using authentication questions. If email notification is enabled and authentication questions has been successfully answered, then an email is sent to the user using the template “
LoginForm	LoginForm adds the errors the AuthFilter encountered to the login page.

In the following example, a view handler and a custom form are specified, but no page processor class is set. (The default `GenericEditForm` class processes this page.)

```
<action path="/spe/user/ForgotUsername"
type="com.sun.idm.idmx.web.ProcessFormAction"
parameter="IDMXLookupUsername,#ID#UserForm:IDMXForgotUsernameForm,">
```

The following example shows an action with a view handler (`IDMXUserQuestion`) and a page processor (`UserQuestionForm`) specified, but no form name was given. The view handler will use the default form:

```
<action path="/spe/user/LoginWithQuestions"
type="com.sun.idm.idmx.web.ProcessFormAction"
parameter="IDMXUserQuestion,,com.sun.idm.idmx.web.UserQuestionForm">
```

Secure Connections

The `secure` parameter forces the server to use HTTPS regardless of the last action's response. Add this parameter as the fourth parameter to force the action to use HTTPS:

```
<action path="/spe/user/Login" type="com.sun.idm.idmx.web.ProcessFormAction"
parameter="IDMXNoop,#ID#UserForm:IDMXLoginForm,com.sun.idm.idmx.web.LoginForm,secure">
        <forward name="success" path=".page.Login"/>
        <forward name="post" path="/spe/user/Login.do" />
        <forward name="next" path="/spe/user/protected/Home.do?newView=true" />
        <forward name="cancel" path="/spe/user/Login.do" />
</action>
```

If your servers do not use port 80 for HTTP or port 443 for HTTPS, modify `WEB-INF/web.xml` to include the following context parameters:

```
<context-param>
    <param-name>listenPort_http</param-name>
    <param-value>7001</param-value>
</context-param>

<context-param>
    <param-name>listenPort_https</param-name>
```

```
    <param-value>7002</param-value>
</context-param>
```

The above code sample tells Struts to use port 7001 for HTTP and port 7002 for HTTPS.

Assigning Navigation Logic

The `forward` elements in the `action` definitions tie the sample pages together. The following table lists the most commonly used forward definitions:

Name	Description
success	Specifies the page to display if there is no error processing the action.
post	Posts the form to the specified location.
next	Specifies the action to take after a successful form posting,.
cancel	Specifies the action to take if a user clicks a cancel button on a form.

SPEUserPages Configuration Object

The following table lists the major attributes in the SPEUserPages configuration object. Edit this object to change when to send notification emails, how to handle password resets, or integrate the user pages with Access Manager or similar product.

Attribute	Description
enrollment	Controls enrollment options. This attribute contains the following subattributes: ■ `validation.enabled`— If `true`, the validation page is displayed and the user must verify his relationship with the provider. The default is `true`. ■ `privacypolicy`— If `true`, the user must accept the privacy policy before completing the registration. The default is `true`.

Attribute	Description
lookup-attributes	A list of attributes that are used to retrieve a user's ID or password. By default, the user's `homephone` and `email` attributes are used, but any attribute defined as an Identity System User Attribute in the schema map for the resource may be used. This attribute contains the following subattributes: ■ `name`— The name of the attribute to use to look up the user. ■ `title`— The message key of the title to display for the lookup user form. ■ `required`— A boolean indicating whether the attribute is required in the lookup user form.
notification	Indicates when an e-mail notification should be sent to the user. This attribute contains the following subattributes. The possible values for these subattributes are `true` and `false`. The notification values are ■ `emailchange` ■ `lockout` ■ `passwordchange` ■ `questionchange` ■ `recovery` ■ `registration` ■ `useridchange`
password	Specifies how password resets should be handled. This attribute contains the `password-reset` subattribute, which must be set to one of the following values:. ■ `self`— The user can reset his or her password if all the challenge questions have been answered correctly ■ `notification`— The user is sent a temporary password to the notification address if all the challenge questions have been answered correctly.
sso-assume-authenticated	If set to `true`, the AuthFilter will not redirect to a login page. However, for auditing purposes, the filter requires a user name to associate with each request. Normally, this username is stored on the HTTP session by the login pages. However, since the login pages are not used in an SSO environment, the username is pulled from an HTTP header attribute.
sso-user-name-http-header-attr	Specifies the name of the HTTP header attribute to use.

The following table lists the default values of the `notification` attribute and its corresponding email template. The email templates can be edited from the Identity Manager Administration

Interface, but the preferred method of changing the subject and body of the email is to modify the messages in the `IDMXMessages.properties` file to the desired text.

Notification Value	Email Template
emailchange	Service Provider End-User Change Notifications
	Service Provider End-User Change Notifications Old Address
lockout	Service Provider End-User Profile Locked
passwordchange	Service Provider End-User Change Password
passwordreset	Service Provider End-User Password Reset
questionchange	Service Provider End-User Authentication Answers Updated
recovery	Service Provider End-User Username Recovery
registration	Service Provider End-User Registration Template
useridchange	Service Provider End-User Change User Id

Note – For the emailchange option, notification is sent to both the new and old e-mail addresses.

Using Apache Struts to Create User Pages

The Service Provider User Interface pages are constructed using the Tiles component of the Apache Struts Framework.

One approach to separating layout from content is the use of dynamic JSP includes. The Tiles framework uses a more advanced form of the JSP include action. In a Tiles application, the layout template usually defines the position of a header, menu, body content, and footer. Other pages are then included to fill each of these positions. Tiles makes using template layouts easier through the use of a simple tag library.

Technically, a tile is a rectangular region in a JSP. A tile may be assembled from other tiles, creating a tree-like hierarchy. A tile can accept variable information at runtime in the form of attributes, making it parameterizable. Tile attributes are defined when inserting the tile and are visible within the tile only. They are not visible in subtiles or to a page enclosing the tile.

Layout Template

In Service Provider, the primary layout template is defined in `$WSHOME/spe/user/common/layouts/page.jsp`.

```
<%@ taglib uri="/tags/struts-tiles" prefix="tiles" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<html:html>
   <head>
      <title><bean:write name="title" filter="off"/></title>
      <html:base/>
   </head>
   <body>
      <tiles:insert name="masthead"/>
      <tiles:insert name="navbar"/>
      <tiles:insert name="subnavbar"/>
      <tiles:insert name="content"/>
      </body>
</html:html>
```

This simple layout template indicates that the body of the page is built from four tiles: `masthead`, `navbar`, `subnavbar`, and `content`. Each tile corresponds to a JSP file in the `$WSHOME/spe/user/common/tiles` directory.

Tiles Definitions

A JSP template system uses one template for the layout and another for the fill-in components. Tiles calls these layout files "definitions". To avoid creating an extra file just to specify the layout, Tiles allow the layout definitions to be stored as a JavaBean. For Service Provider, these definitions are declared in the following XML configuration files, located in the `$WSHOME/WEB-INF/spe/config` directory:

- `tiles-page-defs.xml`
- `tiles-nav-defs.xml`
- `tiles-tab-defs.xml`

These definitions files are specified in the deployment descriptor file, `$WSHOME/WEB-INF/web.xml`:

```
<init-param>
   <param-name>definitions-config</param-name>
   <param-value>/WEB-INF/spe/config/tiles-tab-defs.xml,
                /WEB-INF/spe/config/tiles-nav-defs.xml,
                /WEB-INF/spe/config/tiles-page-defs.xml</param-value>
</init-param>
```

The following example shows the base definition for a Service Provider page, as defined in `tiles-page-defs.xml`:

```
<definition name=".page.Base" path="spe/user/common/layouts/page.jsp"
      controllerClass="com.sun.idm.idmx.web.LocalizePageTitles">
```

```
   <put name="page_title_key" value="${page_title_key}" />
   <put name="masthead" value="spe/user/common/tiles/masthead.jsp" />
   <put name="navbar"   value="${navbar}" />
   <put name="left_nav" value="spe/user/common/tiles/left_nav.jsp" />
   <put name="content"  value="${content}" />
   <put name="footer"   value="spe/user/common/tiles/footer.jsp" />
</definition>
```

The attributes passed into this definition may consist of message resource bundle keys, like `page_title_key`, or references (directly or indirectly) to other tiles inserted on the page.

A tile definition may include a controller class. This class may access tile attributes from the context before the tile is rendered. In Service Provider, the controller class looks up the appropriate page title or subtitle from the message resource bundle based on the key passed as a tile attribute. The Struts framework uses a locale-specific resource bundle, if it exists.

Adding a New Page

The following example extends the Base definition and uses a custom JSP named `my_page.jsp` to render the content tile. It renders content appropriate for the for a JSP named `my_page.jsp`:

```
<definition name=".page.MyPage" extends=".page.Base">
   <put name="page_title_key" value="message.my_page_title" />
   <put name="navbar" value=".nav_bar.MyPage" />
   <put name="content" value="/pages/my_page.jsp" />
</definition>
```

The rendered page consists of content from the masthead, navbar and content tiles. The content tile contains the output from `my_page.jsp`. (Even though the Base page tile definition contains references to the left-hand and “footer” tiles, these do not appear on the rendered page because they are not included in the `page.jsp` layout template.) The localized title string will be rendered in the browser’s title bar.

The definition name (`.page.MyPage`) can be referenced in a Struts ActionForward in the same manner as a regular file URI. Slashes can be used in a definition name, but by best practice convention, a period is used as a delimiter to distinguish the tiles definitions from file URIs, such as "`/spe/user/Login.do`".

Updating the Navigation Bar

Once you have defined a new page to be added to the Service Provider User Interface, you must add a button to the navigation bar so that users can access the new page.

The navigation and subnavigation bars are implemented with tab-style buttons created from tiles. The JSP for a tab button outputs an HTML fragment consisting of a single table cell. The cell contains the localized text and optional hyperlink for the tab.

The following example lists the base definition for an unselected navigation button:

```
<definition name=".nav_tab.Base"       path="spe/user/common/tiles/nav_tab.jsp"
      controllerClass="com.sun.idm.idmx.web.LocalizeButtonLabel">
   <put name="button_link" value="${button_link}" />
   <put name="button_label_key" value="${button_label_key}" />
   <put name="link_class" value="Tab1Lnk" />
</definition>
```

The first two attributes are the message resource bundle keys associated with the text displayed within the button and the URI of the hyperlink for that text. The remaining attributes are the style sheet classes that give the button its appearance.

The base definition for a selected navigation button requires different style sheet classes. These classes must specify a different background color to indicate the tab has been selected and deactivate the hyperlink in the button text.

The following example lists the base definition for a selected navigation button:

```
<definition name=".nav_tab_selected.Base"       path="spe/user/common/tiles/nav_tab_selected.jsp"
      controllerClass="com.sun.idm.idmx.web.LocalizeButtonLabel">
   <put name="button_label_key" value="${button_label_key}" />
   <put name="text_class" value="Tab1SelTxtNew" />
   <put name="button_class" value="Tab1TblSelTd" />
</definition>
```

To create the definitions for the new button in each state, extending the base definitions as follows:

```
<definition name=".nav_tab_selected.MyPage"
       extends=".nav_tab_selected.Base">
   <put name="button_text" value="My Page" />
</definition>
<definition name=".nav_tab.MyPage" extends=".nav_tab.Base">
   <put name="button_link" value="/spe/user/MyPage.do" />
   <put name="button_text" value="My Page" />
</definition>
```

The individual button tiles can be combined together into a new tile for the navigation bar itself. To do this, pass a list of button tiles to the following base navigation bar definition:

```
<definition name=".nav_bar.Base"
       path="spe/user/common/tiles/nav_bar.jsp">
   <put name="div_class" value="Tab1Div" />
   <put name="context_class" value="Tab1TblNew" />
</definition>
```

Use the following example to define a list of buttons that will appear in the navigation bar for the new page:

```
<definition name=".nav_bar.MyPage" extends=".nav_bar.Base">
   <putList name="tab_list" >
      <add value=".nav_tab.Home" />
      <add value=".nav_tab.Profile" />
      <add value=".nav_tab_selected.MyPage" />
   </putList>
</definition>
```

The JSP that renders the navigation bar (`spe/user/common/tiles/nav_bar.jsp`) loops through this list of buttons and includes the appropriate tile, passing attributes specific to that button. (The attributes can include the hyperlink, button text, style class, etc.).

To force the new button to appear in the navigation bar on other pages, you must add the (non-selected) definition to each page-specific navigation bar definition. For example, to add our new button to the navigation bar on the home page:

```
<definition name=".nav_bar.Home" extends=".nav_bar.Base">
   <putList name="tab_list" >
      <add value=".nav_tab_selected.Home" />
      <add value=".nav_tab.Profile" />
      <add value=".nav_tab.MyPage" />
   </putList>
</definition>
```

Tag Library

The following custom tags may be used in the user pages:

`counter`— A counter that is incremented by one every time the tag is called during a request. This tag is only used in the masthead, navigation bar, and the navigation button tiles. You may specify the value attribute to set the initial value of the counter. For example, to reset the counter, `counter value="1"`.

`generateForm`— Displays the content produced by the page processor. This tag may be used in content-based tiles, such as those used in `Form.jsp`.

Tracing Struts Messages

Messages from Struts can be logged by enabling the Identity Manager trace feature and specifying trace levels for the `org.apache.struts` package, or any of its subclasses/subpackages. Struts logging does not allow configuring the names of specific methods that should be logged, and this level of configuration will be ignored.

Changing the Appearance of Service Provider End User Pages

The Apache Struts Action framework provides navigation between the pages, while the Struts/Tiles templating framework is used to create common look and feel for the application.

Modifying the Layout

The Service Provider User Interface pages implement the Apache Struts Action framework and Tiles to control page layout. The file `$WSHOME/spe/user/common/layouts/page.jsp` is used as a template when pages are rendered, with various regions of the page corresponding to the output of individual JSPs or tiles. The default page template consists of four tiles:

- masthead
- navbar
- subnavbar
- content

Adding a new tile, for example a "footer" at the bottom of the page, is accomplished by defining the tile in `$WSHOME/WEB-INF/spe/config/tiles-page-defs.xml` and then inserting the tile in `page.jsp`:

```
<body class="DefBdy" onload='checkFirst( );'>
   <tiles:insert attribute="masthead" ignore="true"/>
   <tiles:insert attribute="navbar" ignore="true"/>
   <tiles:insert attribute="subnavbar" ignore="true"/>
   <tiles:insert attribute="content"/>
   <tiles:insert attribute=”footer” ignore=”true”/>
</body>
```

Placement of the tile on the rendered page can be controlled via HTML markup in the page template and/or via CSS. A template tile for a page footer is located at `$WSHOME/spe/user/common/tiles/footer.jsp`.

Another page which users will frequently want to modify is the login page. This page consists of a single tile, `content`, whose source JSP is `$WSHOME/spe/user/Login.jsp`. The actual HTML `<FORM>` is generated as an Identity Manager form, but it is wrapped in content from `Login.jsp`.

The `page.jsp` file includes three CSS stylesheet files:

- `styles/lockhart.css`
- `styles/style.css`
- `styles/customStyle.css`.

The first two stylesheets are the system defaults and should not be modified, as they may be rewritten during upgrades. The `customStyle.css` stylesheet is empty and is intended for user custom CSS styles. These files are all located in `$WSHOME/spe/user/common/layouts/styles/`.

The Service Provider User pages are designed so that rebranding the page can be accomplished by modifying CSS styles, rather than JSPs. For example, the product name logo (Java System Identity Manager Service Provider) image in the masthead is specified by the following CSS definition:

```
td.MstTdTtlProdNam {
   background: url(../images/ProductName.png) no-repeat 10px 30px;
   padding:10px 10px 0px 10px;
   vertical-align:top;
   white-space:nowrap;
   height: 75px;
   width: 520px;
}
```

You can override this definition in `customStyle.css` to point to a different image file or specify a different location. For more information about using CSS to change user interface elements in Identity Manager, see Deployment Guide.

Internationalization and Localization

Service Provider uses a standard Java message resource bundle for display strings in the UI. This resource bundle is normally included in the `idmspe.jar` file and must be extracted if you plan to customize message strings for display in the default en_US locale.

```
$ cd $WSHOME/WEB-INF/classes
$ jar xvf ../lib/idmspe.jar com/sun/idm/idmx/msgcat/IDMXMessages.properties
```

The format of the resource bundle is the standard *key* =*value* format. In the example `page.title.home=Home` , `page.title.home` is the key and `Home` is the value associated with the key. The value string may be parameterized as standard `java.text.MessageFormat` strings.

There are two types of message keys in the resource bundle: those used by Struts/Tiles and those used by the Identity Manager forms engine. Keys recognized by Struts/Tiles will NOT be recognized by the Identity Manager forms engine, and vice versa.

Struts/Tiles message keys use the “dot notation” format, such as `page.title.home`. Message keys for the Identity Manager forms engine and for email notification templates must contain an underscore character and no spaces, such as `IDMX_login_field_username`. For more information on using the Identity Manager forms engine keys, see Deployment Guide.

For locales other than en_US, the resource bundle file contains the same key values as the default file and appropriate value string translations. For example, the message resource bundle strings for the fr_FR locale would go in `$WSHOME/WEB-INF/classes/com/sun/idm/idmx/msgcat/IDMXMessages_fr_FR.properties` .

Localization and Email Templates

Message bundle keys may also be used in email templates. The default Service Provider End-User email templates are stored in the repository and have IDs similar to `#ID#EmailTemplate:SPEEndUserChangePassword` . Message bundle keys may be specified for the email subject and body of the message:

```
<!-- The template used when user changes their password. -->
<EmailTemplate id='#ID#EmailTemplate:SPEEndUserChangePassword'
name='Service Provider End-User Change Password'
displayName='UI_EMAILTEMPLATE_SPE_END_USER_CHANGE_PASSWORD'
smtpHost='$(smtpHost)' fromAddress='admin@example.com'>

   <subject>IDMX_change_password_email_subject</subject>
   <body>IDMX_change_password_email_body</body>
   <MemberObjectGroups>
      <ObjectRef type='ObjectGroup' name='All'/>
   </MemberObjectGroups>
</EmailTemplate>
```

Note that message bundle keys used in email templates MUST contain an underscore character to be recognized as a message key.

Localizing Strings in the Navigation Bars and Page Titles

Strings that appear in the horizontal navigation bars and browser title are handled by the Struts framework and are referred to by message keys in the resource bundle which use the "dot notation" naming convention.

For example, to change the subnavigation tab for "Password" (under the "My Profile" tab) to use the string "My Password" we must first look up the message bundle key. This is defined in `$WSHOME/WEB-INF/spe/config/tiles-tab-defs.xml` :

```
<!-- Unselected sub-navigation tab for changing the user's password -->
<definition name=".subnav_tab.Password" extends=".subnav_tab.Base">
   <put name="button_link" value="/spe/user/protected/ChangePassword.do?newView=true" />
   <put name="button_label_key" value="nav.tab.label.password" />
</definition>
```

In this case, the message bundle key `nav.tab.label.password` is used for the (en_US) string "Password" which will appear as the tab label text. See above for how to extract and modify the message resource bundle file.

Note that the above Tile definition extends a definition called `.subnav_tab.Base` , which is defined as follows:

```
<!-- Base type for unselected sub-navigation tabs -->
<definition name=".subnav_tab.Base" path="/spe/user/common/tiles/subnav_tab.jsp"
controllerClass="com.sun.idm.idmx.web.LocalizeButtonLabel">
   <put name="button_link" value="${button_link}" />
   <put name="button_label_key" value="${button_label_key}" />
   <put name="link_class" value="Tab2Lnk" />
</definition>
```

We are passing the parameter `${button_label_key}` a specific value of "`nav.tab.label.password`" in the `.subnav_tab.Password` definition. The base tile definition has a controller class associated with it called `com.sun.idm.idmx.web.LocalizeButtonLabel`. The purpose of this controller class is to take the value of parameters passed to it, like `button_label_key`, look them up in the message bundle and put the resulting string back into the Tiles context so that they may be referred to in the rendering Tile. The JSP which renders an unselected subnavigation bar tab, `subnav_tab.jsp`, looks like:

```
<%@ taglib uri="/tags/struts-tiles" prefix="tiles" %>
<%@ taglib uri="/tags/spe" prefix="spe" %>
<tiles:useAttribute name="button_link"/>
<tiles:useAttribute name="button_text"/>
<tiles:useAttribute name="link_class"/>
<td><a href="<%=request.getContextPath()+button_link%>" tabindex="<spe:counter/>"
class="<%=link_class%>"><%=button_text%></a></td>
```

Note that the controller puts the message bundle string back into the Tiles context as the attribute `button_text`.

A similar technique is used to define the strings which will appear as titles and subtitles on pages which are simple Struts forms (as opposed to a "form" created by the Identity Manager forms engine). An example of a Struts form is the OperationResult page. This page displays the results of an operation like changing a user's password or notification address. A controller class called `com.sun.idm.idmx.web.LocalizePageTitles` is associated with the base tile definition for this page. This controller class may be passed message bundle keys for the page title:

```
<!-- Page showing success message after a user resets the password. -->
<definition name=".page.ResetPasswordSuccess" extends=".page.OperationResultNoAuth">
   <put name="forward_id" value="ResetPasswordSuccess" />
   <put name="page_title_key" value="page.title.password_reset_success" />
   <put name="page_subtitle_key" value="page.subtitle.password_reset_success" />
</definition>
```

OperationResult Relay Definition

When a user resets a password or changes the user ID in Service Provider, the results of the operation are displayed to the user. Instead of creating a separate JSP for each successful operation, it is desirable to have a generic OperationResult page that contains the following:

- The title and subtitle of the page
- The appropriate information that indicates the user's change was successfully applied to the directory
- The page (URL) should the user go to after acknowledging the "success" message.

The title and subtitle strings are inserted into the page `$WSHOME/spe/user/OperationResult.jsp` . We would also like to be able to pass a key from the tile definition to look up the appropriate action forward for the user's final destination.

```
<put name="forward_id" value="ResetPasswordSuccess" />
```

When the `.page.ResetPasswordSuccess` tile is rendered, the `forward_id` parameter is added to the form as a hidden field with name `dispatch`.

When the user acknowledges the result by clicking the **OK** button, the following action is executed:

```
<!-- Set action forward mapping to final destination forward -->
<action path="/spe/user/OperationResult" name="resultForm"
      type="com.sun.idm.idmx.web.RelayAction" scope="request"
      validate="false">
   <forward name="ForgotUsernameSuccess" path="/spe/user/Login.do?newView=true"
      redirect="true"/>
   <forward name="PasswordEmailed" path="/spe/user/Login.do?newView=true"
      redirect="true"/>
   <forward name="ResetPasswordSuccess" path="/spe/user/Login.do?newView=true"
      redirect="true"/>
   <forward name="ProfileLocked" path="/spe/user/Login.do?newView=true"
      redirect="true"/>
   <forward name="ProfileHasBeenLocked" path="/spe/user/Login.do?newView=true"
      redirect="true"/>
   <forward name="EnrollSuccess" path="/spe/user/Login.do?newView=true"
      redirect="true"/>
</action>
```

The `com.sun.idm.idmx.web.RelayAction` class looks up the `dispatch` request parameter value in the above definition and forwards to the associated URI. In our case, `ResetPasswordSuccess` maps to the URI "/.m"`/spe/user/Login.do?newView=true`".

Error Handling

In general, errors from which the user can recover, such as form validation errors, are displayed on the edit form so the user can correct the data and resubmit the form.

Errors encountered during processing are generally not recoverable and must be handled differently. A generic error action forward, `spe/user/Error.do` , is used to display a message

that the user should report the error to their local administrator. The JSP supporting this action, `$WSHOME/spe/user/Error.jsp` , will also include any exception stack trace information in the rendered page source as an HTML comment.

Index

W