

## Chapter 2

# Installing and Configuring the Sample Residential Portal

This chapter provides an overview of the sample residential portal, explains how to install and configure it, and describes how to create a portal based on the sample. The chapter contains the following sections:

- Overview of the Sample Residential Portal on page 21
- Authenticating Subscribers Subscriber Login Through RADIUS on page 24
- Installing the Sample Residential Portal on page 24
- Developing a Portal Based on the Sample Residential Portal on page 33

## Overview of the Sample Residential Portal

---

The sample residential portal is a demonstration portal that shows how to use some of the features available in the Common Object Request Broker Architecture (CORBA) remote application programming interface (API) to create a Web application. You can customize the sample residential portal for your environment, or create a new Web application using the SAE CORBA remote API.

## Web Application Architecture

The sample residential portal uses the Jakarta Struts Web application framework. Although Struts provides an easy and extensible framework for building Web applications, it is not required for building portals that use the CORBA remote API.

Jakarta Struts supports the model-view-control design paradigm, which separates an application into three sets of components:

- Model—Contains the data and business logic.
- View—Contains the presentation to the subscriber.
- Control—Contains the interface procedures.

The strict separation of the three layers promotes reuse of the components and allows easy adaptation of the application to different requirements.

## Model Components

The model provides an abstraction layer of the CORBA remote API and contains the business logic, which determines how the sample portal behaves. The sample residential portal includes two implementations of the model (which we call behaviors) to demonstrate some typical usage scenarios. See *Behaviors for the Sample Residential Portal* on page 22 for more information.

## View Components

The view components of the Web application provide the HTML code sent to the subscriber's browser. The view is implemented by means of JavaServer Pages (JSP) and several tag libraries provided as part of Jakarta Struts.

The tiles tag library provides a template mechanism to build Web pages based on reusable partial pages. The general layout of all pages of the portal application is defined in a single JSP page.

## Control Components

The control components provide the interactions between the subscriber and the mode through the Action and ActionForm classes.

Action classes implement the functionality for a single operation, such as “list the subscriptions of a particular service category,” or “activate a service.”

ActionForm classes encapsulate data provided by the subscriber on an input form. The Struts framework initializes these classes with data entered in an HTML form and passes them to the appropriate action. The ActionForms are then passed to a view component that uses the data to initialize the content of fields in an input form.

## Behaviors for the Sample Residential Portal

The sample residential portal provides two user behaviors (scenarios) that integrate with the Merit RADIUS server:

- Equipment registration
  - Used by subscribers who use Dynamic Host Configuration Protocol (DHCP) connections to register their devices to receive an authenticated IP address.
- Internet Service Provider (ISP) service
  - Used by subscribers who use Point-to-Point Protocol (PPP), static IP, or unauthenticated DHCP connections to log in to the portal and receive an unauthenticated IP address.



**NOTE:** By default, the sample residential portal sends unrecognized IP subscribers to a login page rather than to an error page.

You can customize how unrecognized IP subscribers are handled in the *struts-config.xml* file.

---

### Equipment Registration Behavior

The equipment registration example demonstrates an application that provides an association between a subscriber and the equipment being used to make the DHCP connection. This type of association is used in many cable environments.

To configure a Merit RADIUS server to demonstrate equipment registration:

1. Move to the *radius/etc* directory.

```
cd /opt/UMC/radius/etc
```

2. Copy *authfile.equipment* to *authfile*.

```
cp authfile.equipment authfile
```

With equipment registration, a subscriber registers equipment to the SAE only once. At registration, the system caches the media access control (MAC) address of the device; thereafter, the MAC address identifies the device to the SAE, and an authenticated IP address is returned to the device.

When the MAC address of the subscriber's equipment is associated with a user profile, the subscriber login can be configured as persistent. With a persistent login, the subscriber does not need to log in again as long as the registered MAC address remains the same. This process lets non-HTML-capable devices, such as IP phones and set-top boxes, to be registered to the network. If subscribers do not have a persistent login configured, the portal detects that the subscriber is not authenticated and directs the subscriber to a login page.

### ISP Service Behavior

The ISP service example demonstrates an application that provides a means for subscribers to directly log in to a subscriber session for their ISP. The ISP service behavior is well suited for any environment in which subscribers connect directly to their ISP.

To configure a Merit RADIUS server to demonstrate the ISP service behavior:

1. Move to the *radius/etc* directory.

```
cd /opt/UMC/radius/etc
```

2. Copy the *authfile.isp* to *authfile*.

```
cp authfile.isp authfile
```

This connection can be to a wireless device or over physical connection media. In this scenario, subscribers can log in to their ISP from any device; there is no need to bind a particular subscriber to a particular PC or workstation (equipment registration).

The ISP service model applies to subscribers who log in to a specific provider. To switch between providers, subscribers can log out from one and then log in to another.

## Authenticating Subscribers Subscriber Login Through RADIUS

---

If you use RADIUS to manage subscriber data, you can use RADIUS to authenticate subscribers when they log in to a residential portal. You configure RADIUS authentication plug-ins to provide RADIUS authentication or authorization. In the configuration for the plug-in, you specify how the SAE handles RADIUS attributes received from the RADIUS server.

Because the SAE rather than a JUNOSe router receives the authentication response, you can specify that the response include attributes other than serviceBundle and class, and you can specify more than value for the RADIUS class attribute.

To authenticate subscribers through RADIUS at portal login:

1. Create a RADIUS authorization plug-in to authenticate subscriber sessions.
2. Configure the RADIUS authorization plug-in to specify:
  - The RADIUS attributes to be set in an authorization response
  - The action to be taken in response to the attribute values received

For example, you could create a RADIUS authorization plug-in to:

- Authenticate a PPP subscriber session on a JUNOSe router
- Specify the setLoadServices value for the serviceBundle attribute

By default, the flexible RADIUS authentication plug-in defines this attribute as:

```
RadiusPacket.stdAuth.userresp.vendor-specific.Juniper.Service-Bundle =
setLoadServices
```

For more information about RADIUS authentication plug-ins, see *SDX Components Guide, Vol. 1, Chapter 5, Configuring Authorization and Accounting Plug-Ins*.

## Installing the Sample Residential Portal

---

The sample residential portal is a Web application. The application is packaged as a standard Web application archive (WAR file) in the *webapp* subdirectory in the SDX software distribution.

To install the sample residential portal:

1. Complete prerequisite tasks.
2. Prepare the Web application for customization.

3. Configure deployment settings, including a connection to the directory.
4. Deploy the Web application.



**NOTE:** The sample residential portal can be installed by root or authorized nonroot users.

---

### Prerequisite Tasks

Before you install the sample residential portal:

- Install a Web application server on the machine on which you want to install the sample residential portal.

We provide the JBoss Web application server in the SDX software distribution. For information about installing this software, see *SDX Software Basics Guide, Chapter 11, Installing Web Applications*.

- Install the sample data from the SDX software distribution (see *SDX Software Basics Guide, Chapter 7, Defining an Initial Configuration*).
- Set up the RADIUS *authfile* for the user scenario you want to demonstrate. See *Behaviors for the Sample Residential Portal* on page 22.

### Preparing the Application for Customization

When you customize the sample residential portal, copy the WAR file to a temporary folder and work in that folder. To do so:

1. Login as `root` or another authorized user.
2. Create a temporary folder in which you will work on the WAR file.

**mkdir ssportal**

3. Access the temporary folder.

**cd ssportal**

4. Copy the WAR file to the temporary folder.

**cp /cdrom/cdrom0/webapp/ssportal.war.**

## Configuring the Sample Residential Portal

To configure the sample residential portal:

1. Access the temporary folder to which you copied the WAR file.

**cd ssportal**

2. Extract the files from the WAR file.

**unzip -quo ssportal.war**

3. With a text editor, edit the *portalBehavior.properties* file and other files in the *WEB-INF* directory as needed. See *Modifying Configuration Files* on page 27.

Use *WEB-INF/portalBehavior.properties* on page 27 as a guideline for editing the *portalBehavior.properties* file to use properties specific to your environment.

4. Replace the *portalBehavior.properties* and any other updated files in the WAR file.

**zip -u ssportal.war**

## Deploying the Updated WAR File

To deploy the updated WAR file:

- Copy the WAR file to the deployment directory for your Web server.

If you are using JBoss, copy the file to */opt/UMC/jboss/server/default/deploy* directory. JBoss automatically starts the Web application when a new WAR file is copied into the deployment directory.

By default the sample residential portal is deployed into the root context (“/”). You can access the portal through *http://server:8080*. If you want to deploy the sample residential portal into something other than the root context, modify the *WEB-INF/jboss-web.xml* configuration file.

## Removing Access to the Sample Residential Portal

To remove access to the sample residential portal:

- Remove the *ssportal.war* file from the deployment directory.

## Modifying Configuration Files

The `ssportal.war` file contains the following configuration files in the `WEB-INF` directory:

- `portalBehavior.properties`—Specifies properties to configure the `portalBehavior` servlet that determines the behavior of the sample residential portal.

Modify this file to run the sample residential portal. See *WEB-INF/portalBehavior.properties* on page 27.

- `web.xml`—Specifies the deployment descriptor for the sample residential portal. It describes the servlets, other components, and initialization parameters.



**NOTE:** We recommend that you do not change the deployment descriptor.

---

- `jboss-web.xml`—Contains one configuration property that defines the Web context of the sample residential portal as the root context.

Modify this file to run the sample residential portal in a context other than root. The `WEB-INF/jboss-web.xml` file is proprietary to the JBoss application server.

- `struts-config.xml`—Contains the configuration for the struts action servlet. See *WEB-INF/struts-config.xml* on page 29.
- `tiles-defs.xml`—Contains the definitions of the tiles template system. The definitions describe the general layout of every Web page used in the sample residential portal. See *WEB-INF/tiles-defs.xml* on page 32.

### WEB-INF/portalBehavior.properties

Set the following properties to configure the `portalBehavior` servlet to determine the behavior of the sample residential portal, and to connect to the LDAP server.

In addition, configure the other properties listed in the file for the network information collector (NIC) proxy configuration. For information about the values to configure for NIC properties, see *Chapter 13, Configuring Applications to Communicate with an SAE*.

### Factory.behavior

- Model for handling subscribers who connect using DHCP.
- Values
  - `net.juniper.smgmt.ssp.model.EquipmentRegistrationBehavior`
  - `net.juniper.smgmt.ssp.model.ISPServiceBehavior`
- Guidelines—For information about the two behaviors, see *Behaviors for the Sample Residential Portal* on page 22.

**Factory.locator**

- Method that the portal uses to locate the SAE that is managing the subscriber who tries to access the application.
- Values
  - `net.juniper.smgmt.ssp.LocalFeatureLocator`—Uses the locally configured object reference  
 If you specify `net.juniper.smgmt.ssp.LocalFeatureLocator`, configure a value for `LocalFeatureLocator.objectRef`.
  - `net.juniper.smgmt.ssp.DistributedFeatureLocator`—Uses NIC configuration

**LocalFeatureLocator.objectRef**

- CORBA object reference for the single SAE whose address is resolved by the locator. Specify the object reference if you set `net.juniper.smgmt.ssp.LocalFeatureLocator` for `Factory.locator`.
- Values—A reference to the CORBA object in one of the following formats:
  - The absolute path to the IOR file in the form `file:// <absolutePath >`
  - The corbaloc URL in the format:  
`corbaloc:: <host > : <port > /SAE`
    - `<host >` — IP address or host on which the SAE is installed.
    - `<port >` —TCP/IP port number for the SAE. The default is 8801.
  - COS naming service in the format:  
`corbaname:: <host > [: <port > ][/NameService]# <key >`  
 where `<key >` is provided by the publisher of the IOR to the COSnaming service.
  - The actual IOR in the form `IOR: <objectReference >`
- Example
  - Absolute path—`file:///opt/UMC/sae/var/run/sae.ior`
  - corbaloc URL—`corbaloc::10.10.6.171:8801/SAE`
  - Actual IOR—  
`IOR:000000000000002438444C3A736D67742E6A756E697...`
- Guidelines—Configure this property to use the portal as a demonstration application in a small environment that does not use NIC.  
 By default, the SAE does not publish its IOR to a COSNaming service.

**LocalFeatureLocator.vrName**

- Virtual router to use in a Packet Cable Multimedia (PCMM) environment as the virtual router on the local machine.
- Value—default@simJunos
- Guidelines—Configure this property only if you configured a value for LocalFeatureLocator.objectRef.

**DistributedFeatureLocator.locName**

- Namespace for the NIC proxy configuration.
- Value— < namespace >
- Default—/ which indicates the root namespace
- Example—DistributedFeatureLocator.locName = /nicProxy indicates that the NIC proxy configuration is in /nicProxy.

**Config.java.naming.provider.url**

- Location of the LDAP server.
- Value—ldap:// < IP address > : < port number >
- Example—ldap://127.0.0.1:389 (default location if you are using the default OpenLDAP installation from the SDX installation).

**Config.net.juniper.sgmt.des.backup\_provider\_urls**

- Location of a backup LDAP server.
- Value—ldap:// < IP address > : < port number >

**WEB-INF/struts-config.xml**

The *WEB-INF/struts-config.xml* file contains the following settings. The file has multiple sections.

**data-sources**

- Not used by the sample residential portal.

**form-beans**

- Holds data entered in an HTML form and makes it available to the associated action.

**global-exceptions**

- Specifies that the sample residential portal declare one global exception handler, which is invoked for any exception raised during action processing.

**global-forwards**

- Global forwards for handling error situations. The sample residential portal declares a number of global forwards.
- Values
  - unknownUser—Used when an action is processed for a subscriber who is not known by the system. The possible pages are either *.error.unknownUser.page*, which displays an error message, or *.login.page*, which asks the user to log in.
  - nonUniqueUser—Used when a request cannot be mapped to a single subscriber session.  
The sample residential portal uses the IP address of the subscriber, preventing this error.
  - unknownService—Used when a request refers to a service that is not loaded by the SAE. This can happen if services are modified while subscribers are connected to the portal.
  - unknownSubscription—Used when a request refers to a service to which the current subscriber is not subscribed.
  - serviceAuthError—Used if authorization for a service is denied; for example, because mutex group restrictions are violated or a plug-in has denied authorization.
  - loginError—Used if login was unsuccessful.
  - saeError—Used for SAE internal errors.
  - error—Used for any other problem.

**action-mappings**

- Actions that each correspond to an interaction of the subscriber with the portal page. The sample residential portal declares a number of actions.
- Values
  - /index—Displays the main page of the portal; collects information about the subscriber requesting the page and forwards it to the *.index.page*.
  - /services—Gets information about the subscribed services and forwards to the *.services.page*.
  - /activate—Checks whether authentication is required and forwards the request either to the *.service.auth.page* or back to the *.services.page*.  
Called when the subscriber wants to activate a service.
  - /deactivate—Forwards the request back to the *.services.page*.  
Called when the subscriber wants to deactivate an active service.
  - /schedules—Gets information about the service schedule. Allows the subscriber to view and change service schedules. The action forwards the request to the *.schedules.page*.
  - /scheduleOperation—Forwards the request back to the *.schedules.page*.  
Called when the subscriber wants to change the service schedule.

- */usage*—Collects statistics for currently active services and forwards them to the *.usage.page*.
- */account*—Allows modification of the `activationTrigger` property of currently subscribed services. After a change of the `activationTrigger` property has been processed, the action forwards subscribers to the *.account.page*.
- */subscribe*—Allows the subscriber to subscribe to and unsubscribe from services. After processing the subscription change, the action forwards subscribers to the *.subscribe.page*.
- */register*—Allows subscribers to register MAC addresses for authenticated DHCP addresses. The action checks whether the subscriber has provided a username and password and forwards the request to the *.register.auth.page* to enter the username and password or to the *.register.page* displaying the currently registered equipment.
- */unregister*—Allows subscribers to remove MAC addresses that are registered for DHCP addresses. The action checks whether the subscriber provided a username and password and forwards the request to the *.unregister.auth.page* to enter the username and password or to the *.unregister.page* displaying the currently registered equipment.
- */login*—Allows the subscriber to log in to the system. If the login causes a switch of the DHCP IP address, the request is forwarded to the *.wait.page*. If the DHCP IP address remains the same after the login, the request is forwarded to the *.index.page*.
- */logout*—Allows the subscriber to log out of the system. If the logout causes a switch of the DHCP IP address, the request is forwarded to the *.wait.page*. If the DHCP IP address remains the same after the login, the request is forwarded to the *.index.page*.
- */wait*—Checks whether the IP address of the current subscriber is authenticated or unauthenticated. If the address is of the wrong type, the request is forwarded to the *.wait.page*, which will renew itself automatically. If the address is of the expected type, the request is forwarded to *.index.page*.
- */accessDenied*—Processes a captive portal request. The request is forwarded only to the *.error.accessDenied.page*.

### **controller**

- Ensures generation of the correct headers for disabling caching of the generated pages.
- Value—`nocache`

**message-resources**

- Base name of the resource bundle. The resource bundle contains message strings in different languages.
- Values
  - `WEB-INF/classes/net/juniper/smgmt/ssp/ApplicationResources.properties`  
The location of the resource file containing messages in English that is shipped with the sample residential portal.
  - `WEB-INF/classes/net/juniper/smgmt/ssp/ApplicationResources_xx.properties`  
where `xx` is the two-letter ISO language code, optionally followed by an underline and the two-letter country code; for example, `en_CA` for English/Canada or `zh_TW` for Chinese/Taiwan.

To create a sample residential portal that supports other languages, translate the messages and store the translated file in the above location.

**plug-in**

- Processes templates.

**WEB-INF/tiles-defs.xml**

The `WEB-INF/tiles-defs.xml` file contains the following settings.

**site.layout**

- Main definition that specifies the general structure of all pages. The layout is based on a common template file, `/layouts/common.jsp`. The definition contains values for template variables shared by all page definitions.
- Values
  - `title`—Common title of all pages.
  - `header`—Page fragment displaying the header section of the pages.
  - `menu`—Page fragment displaying the menu bar.
  - `footer`—Page fragment displaying the footer section of the pages.
  - `body`—Page fragment displaying the content of the pages. The default setting is empty and should be overwritten by individual page definitions.
  - `color`—Color scheme used by pages. A color scheme consists of a style sheet (`style_sheets/color.css`) and a set of images (stored in `images/color`). The predefined color schemes are blue and green.
  - `menuTag`—Action name of the current page. The menu bar code uses this tag to highlight the action associated with the current page.

**site.layout.nomenu**

- Provides an extension of the main layout that defines a version of the page without a menu bar.

**.\*.page**

- Provides the definition of portal pages. These pages are used for forwards in the action-mappings section of the *struts-config.xml* file. The page definitions extend one of the common layouts and define the value of the body variable as appropriate.

## Developing a Portal Based on the Sample Residential Portal

---

The source code is included with the sample residential portal. To modify the behavior of the portal beyond a simple configuration, install a Java development environment. You can find the source code of the sample residential portal in the directory *WEB-INF/src*. The portal pages are stored in the layout and tiles directories.

The sample residential portal does not require any specific environment, but the procedures below assume that you use the Eclipse platform. A servlet container is required to run the portals during development. We recommend that you use Tomcat and its Eclipse plug-in.

For information about your development environment, see the documentation for the product you are using.

### Preparing to Develop a Portal Based on the Sample Residential Portal

The following instructions describe how to set up a development environment that uses Eclipse and Tomcat on a Solaris platform. If you want to use Eclipse and Tomcat on a different operating system, see the following Web sites:

- For Eclipse:

<http://www.eclipse.org>

- For Tomcat:

<http://jakarta.apache.org/tomcat>

To get ready to develop a portal based on the sample residential portal:

1. Download and install Eclipse from

<http://www.eclipse.org>

2. Download the Tomcat plug-in for Eclipse from

<http://www.sysdeo.com/eclipse/tomcatPlugin.html>

3. Unzip the plug-in into the Eclipse installation directory.

4. Download Tomcat from

<http://jakarta.apache.org/tomcat>

5. Install Tomcat:

```
mkdir $HOME/eclipse
cd $HOME/eclipse
unzip /tmp/eclipse-SDK-2.0.2-solaris-motif.zip
unzip /tmp/tomcatPluginV201.zip
cd $HOME
gzip -dc /tmp/tomcat-4.1.18.tar.gz | tar xvf -
```

6. Start Eclipse.
7. Configure the Tomcat plug-in.

Select Window > Preferences > Tomcat, and configure the Tomcat version and the path where you installed Tomcat.

### **Creating a Portal Project**

To create a new Tomcat project inside Eclipse:

1. Select File > New > Project > Java > Tomcat Project, enter the name of the project, and click Finish.
2. Select File > Import... > Zip File, enter the path for *ssportal.war*; and click Finish.
3. Select File > Properties > Java Build Path > Libraries > Add Jars, open the sample project, navigate to *WEB-INF/lib*, and select all JAR files in the *WEB-INF/lib* directory.
4. Select File > Properties > Tomcat, and click Can update server.xml file.

### **Building the Portal**

Eclipse automatically rebuilds the project when you save a modified source file.

To test or debug the project, run the code inside Tomcat.

To start Tomcat:

- Select Tomcat > Start Tomcat.

You can set break points in your code to debug the code.

## ***Deploying the Portal***

To create a new Web application, set the name of the target WAR file.

1. Select File > Properties > Tomcat.
2. Enter the path of the target WAR file in the field WAR file for export.
3. Right-click the portal project, and select Tomcat Project > Export to the WAR file set in project properties.
4. Copy the WAR file to the final deployment location; for example, */opt/UMC/jboss/server/default/deploy* on your portal server.

