CHAPTER 3

Advanced SESM Customization

This chapter explains some of the advanced customization techniques that you can use with a Cisco SESM web application. In some SESM deployments, it may be possible for the developer to use one of the sample web applications as a starting point and accomplish look-and-feel modifications to the JSP pages without changing the SESM web application's preprogrammed functionality. The deployment descriptor file (web.xml) for the Cisco SESM web application allows the deployer to configure some of the web application functionality without coding changes to the JSP pages and without adding to the JSP pages.

Other SESM deployments will require some JSP-page coding changes or additions to meet the requirements of a specific deployment. You can accomplish some of the advanced customizations by modifying the web.xml for the SESM web application. You accomplish other advanced customizations by modifying existing JSP pages or creating new ones. No Java servlet programming is needed.

The advanced SESM customization techniques fall into two general categories:

- **Decoration of a user shape**—The user-shape decoration mechanisms allow you to create an SESM web application that renders the web-portal user interface based on subscriber characteristics, such as the device, brand, and locale. These sections describe the mechanisms that provide for decoration of the user shape:
  - User Shapes and User-Shape Decoration, page 3-7
  - Using a Sparse-Tree Directory Structure, page 3-9
  - Decorating a User Shape, page 3-16
- **Modifications of SESM web application functionality**—The sample SESM web applications are fully functional and ready for styling, configuration, and deployment. However, in some deployments, modifications to the functionality of an SESM web application may be required. These sections provide you with an overview of the SESM web application functional components and give you some guidance on how to use and modify components:
  - SESM Architecture: An Overview, page 3-2
  - SESM Software Concepts, page 3-6
  - Modifying SESM Web Application Functionality, page 3-25

For illustration purposes, the explanations in this chapter use the NWSP sample web application. Though certain advanced techniques are typically used in an SESM web application, the developer decides what techniques to use, what techniques to modify, and what techniques not to use based on the application's presentation and business requirements.

SESM Architecture: An Overview

The architecture of an SESM web application uses the Model-View-Control (MVC) design pattern.

- Model—Service, subscriber, and policy information in a data repository as well as the operations that can be used to access and modify this data.
- Views—JSP pages that generate the markup language, such as HTML or WML, which determines the user interface.
- Control—Java servlets that process HTTP requests and are responsible for creating any JavaBeans or other objects used by the JSP pages. Each control servlet forwards to a JSP page (a view).

The MVC design pattern allows the processing logic in the Java servlets to be separated from the presentation components in the JSP pages.

The service-provider developer makes deployment-specific modifications, such as look-and-feel customizations and functionality modifications, to the JSP pages that act as views. The SESM model and controls are preprogrammed and configurable by the deployer. No coding tasks related to the model or controls are required to develop an SESM web application.

Model

In an SESM web application, the *model* is responsible for the interactions between a number of system components, possibly including one or more Service Selection Gateways (SSGs), RADIUS-DESS Proxy servers, and data repositories, either a RADIUS AAA server or an LDAP-compliant directory. The model also includes the programming interfaces that an SESM web application uses to interact with these system components and access information in the data repositories.

The SESM software that implements the model is preprogrammed and configurable by the deployer. The service-provider developer is not required to modify any of the model's preprogrammed software.

Controls

A *control* is a Java servlet that prepares an HTTP request for a view. A control is responsible for creating any JavaBeans or other objects (for example, request or session attributes) used by the JSP pages. SESM web application controls are subclasses of the `com.cisco.sesm.navigator.Control` abstract class.

If a control needs to give a JSP page (the corresponding view) access to dynamic data, such as subscriber account data, that the control has retrieved from the model, the control creates a JavaBean and sets the values of the bean properties. The properties hold the data that the control retrieved. In general, the component-to-component flow is that, after creating the bean and setting its properties, the control forwards the request to the corresponding view. The view then uses the bean to access the retrieved data.

After a control has processed an HTTP request, it usually handles a `GET` or `POST` request as follows:

- If the request is a `GET`, the control forwards the request to the corresponding view.
- If the request is a `POST`, the control redirects to a `GET` request for the same control servlet. Redirecting to a `GET` removes the `POST` request from the HTTP client's history list. If someone goes back in the browser history list, the `POST` request and its data will not be available.

Internationalized Resources

The SESM controls provide internationalized data to the views. The JavaBeans created by the SESM controls use internationalized resources, which can be adapted for various languages and regions without programming changes. A view JSP page retrieves these internationalized resources from the view bean. The resource is usually text on a label or button, or a message to the subscriber. For example, the `AccountLogonControl`, a control for subscriber logon page, uses internationalized resources for certain phrases, such as "Please enter your user name." Each text string is a key-value pair in the resource bundle for the SESM web application.

The control internationalizes the resources by associating them with a key. It is the responsibility of the view JSP page to localize the resources. The localization performed by the view JSP page formats the internationalized data (for example, a number or date) according to the current localization context (`L10nContext`). The view JSP pages use the `format` tag of the Localization tag library to perform the localization.

Tip The NWSP web application has the needed code to localize the resources provided by the SESM controls. NWSP software automatically detects the language and country of the subscriber as defined by the subscriber's browser preferences. The NWSP web application has default localization context initialization parameters that the deployer can configure in the web.xml file. For information on the default localization context parameters, see the "Setting a Default Localization Context" section on page 5-12.

Views

When the SESM user-shape mechanisms are used, each JSP page that acts as a *view* for a control provides content targeted for the subscriber's specific characteristics. For example, a view might be implemented in three different JSP pages. Each JSP page provides content using a different markup language, such as HTML, WML and XML, targeted for a different device. Ideally, a view contains little or no Java code so that no Java coding is required.

A view is given all the dynamic data it requires in a JavaBean. Each view is responsible for retrieving data from any beans or other objects that the control creates. The view JSP page uses standard JSP tags to retrieve the data from the view bean's properties. The view can also collect data from the subscriber and post it to a control.

A view JSP page can contain navigation buttons that link to other controls. When the subscriber clicks the button, the HTTP request is sent to the control, which processes the needed information and forwards the request to the view JSP page associated with the control.

Virtual File Names for Views

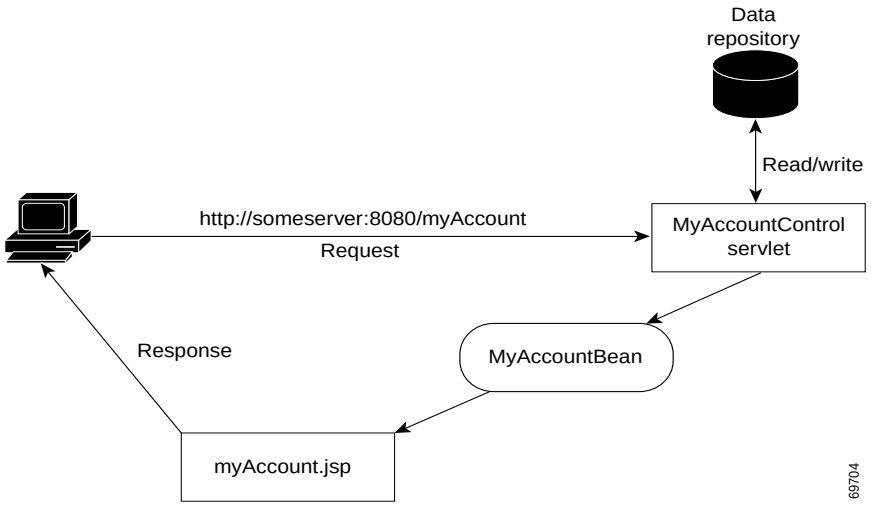
Each control can have many views. For example, there is one `MyAccountControl` but multiple JSP files can be used for the view myAccount.jsp. Each of the JSP pages for myAccount.jsp might provide content tailored for a different device (for example, WAP phone, PDA, or PC). In effect, the `MyAccountControl` forwards to a logical view. The control does not know the actual file name—a URI—for the JSP page and forwards the HTTP request to a *virtual file name*. The control does not know whether there is more than one view or which view will be used. When the control forwards the HTTP request to the logical view, the SESM web-application software uses the characteristics of the subscriber and the subscriber's HTTP client to determine the specific JSP page to use for myAccount.jsp.

The `VirtualFile` servlet translates the virtual file name for the view to an actual file name (URI) based on the subscriber's characteristics. For information on `VirtualFile` and this translation process, see the "Mapping a Virtual File Name to an Actual File Name" section on page 3-33.

MVC Design Pattern Example

Figure 3-1 shows how the NWSP web application uses the MVC design pattern for its My Account page. In LDAP mode, the My Account page (shown in Figure 4-1 on page 4-3) allows the subscriber to display and update account information, such as the subscriber name and address. For simplicity, Figure 3-1 and the accompanying explanation do not show an SSG or explain its role in routing a request.

Figure 3-1 MVC Design for My Account Page



When the subscriber clicks the My Account button in the SESM navigation bar, the following sequence of interactions occurs between the MVC components. The MVC component (Model, View, or Control) that is involved in the interaction is listed before each action.

1. **Control:** The following HTTP request is sent to an SESM web application:

   http://someserver:8080/myAccount

   In the web.xml file, the `MyAccount` servlet name is associated with the `MyAccountControl` servlet class, which is the control for the My Account page.

2. **Control and Model**: The `MyAccountControl` servlet does one of the following:

   - If the request is a `POST`, `MyAccountControl` attempts to use form data to update the subscriber profile in the data repository, adds a `MyAccountBean` to the HTTP session, and redirects to a `GET` request for `MyAccountControl`.
   - If the request is a `GET`, `MyAccountControl` retrieves the account information from the subscriber profile in the data repository, uses an existing or adds a new `MyAccountBean` to the HTTP request, and forwards the request to the myAccount.jsp view.

   In both of the preceding cases, `MyAccountBean` contains the values of the fields (such as names and addresses) that `MyAccountControl` has retrieved from the data repository and that MyAccount.jsp will display.

3. **View**: When the control forwards the request to the view, the myAccount.jsp page (the view) gets user-account properties from `MyAccountBean` for each field in the form. For example, to get the value of the `givenName` field for the form, myAccount.jsp uses `MyAccountBean` to get the value of the `givenName` bean property.
4. **View**: After myAccount.jsp retrieves all user-account information from the bean, it sends the My Account page to the HTTP client, which displays the page.

NWSP Controls, View Beans, and Views

Table 3-1 lists the SESM controls, view beans, and view JSP pages that the NWSP web application uses. The components are listed according to the functionality they provide. The name for a view JSP page corresponds to the control's name with the `Control` part omitted. For example, accountLogoff.jsp is the view that corresponds to the control `AccountLogoffControl`.

The other sample SESM web applications like PDA and WAP provide subsets of the NWSP functionality and use subsets of these controls and beans. They may also use different view JSP pages.

Table 3-1 SESM Web Application Controls, View Beans, and Views

Control	View Bean	View JSP Pages
Account Logon and Logoff		
`AccountLogoffControl`	None	accountLogoff.jsp accountLogoffBody.jsp
`AccountLogon3KeyControl`	`Authenticate3KeyBean`	accountLogon3Key.jsp accountLogon3KeyBody.jsp
`AccountLogonControl`	`AuthenticateBean`	accountLogon.jsp accountLogonBody.jsp
Account Passwords		
`AccountPasswordControl`	None	accountPassword.jsp accountPasswordBody.jsp
SESM Messages		
`MessagesControl`	`MessagesBean`	messages.jsp messagesBody.jsp
Account Information		
`MyAccountControl`	`MyAccountBean`	myAccount.jsp myAccountBody.jsp
Service Subscription		
`SubscriptionConfirmControl`	None	subscriptionConfirm.jsp subscriptionConfirmBody.jsp
`SubscriptionManageControl`	`SubscriptionManageBean`	subscriptionManage.jsp subscriptionManageBody.jsp

Table 3-1 SESM Web Application Controls, View Beans, and Views (continued)

Control	View Bean	View JSP Pages
Service Selection and Logon		

`ServiceListControl`	`ServiceBean`
	`ServiceGroupBean`
	`ServiceListBean`
	`ServiceListServiceBean`
	`ServiceListServiceGroupBean`
		serviceListHead.jsp
		serviceList.jsp
		serviceListService.jsp
		serviceListGroup.jsp

`ServiceLogonControl`	`ServiceAuthenticateBean`	serviceLogon.jsp serviceLogonBody.jsp
`ServiceStartControl`	None	serviceStart.jsp
`ServiceStopControl`	None	serviceStop.jsp
Service Status		
`StatusControl`	`StatusBean`	status.jsp statusBody.jsp
Subaccounts		
`SubaccountConfirmControl`	`SubscriptionManageBean`	subaccountConfirm.jsp subaccountConfirmBody.jsp
`SubAccountListControl`	`SubAccountListBean` `SubAccountDetailBean`	subAccountList.jsp subaccountListBody.jsp
`SubaccountSubscriptionsControl`	`SubaccountSubscriptionsBean`	subaccountSubscriptions.jsp subaccountSubscriptionsBody.jsp

For more information on each control and JavaBean, see the Javadoc for the associated class. The Javadoc for the controls and beans provides information that is useful if you need to modify the code in a view JSP page. For example:

- If a JSP page associated with a control displays a form that the subscriber uses to fill in information, the description of the control includes the `POST` parameters that are expected by the control.
- If a JSP page retrieves information from a JavaBean that a control creates, the description of the bean lists the information that the bean provides.

When a JSP page posts parameters to a control, the HTTP protocol provides no mechanism for checking that the correct parameters and appropriate values have been posted. The posting of parameters requires that the developer test to verify the results.

SESM Software Concepts

This section provides introductory information on some concepts that you need to understand before developing a Cisco SESM web application:

- User Shapes and User-Shape Decoration
- Decorators

User Shapes and User-Shape Decoration

A *user shape* is a set of characteristics that defines the web resources that a Cisco SESM web application uses for a specific subscriber. A `Shape` object encapsulates the set of characteristics that define the web resources available for a specific subscriber. The shape of a user can include characteristics such as the following:

- Devices and browser software used to connect to the web site
- Branding for the web site, such as a brand for business use and a brand for personal use
- Language and country of the user
- Personal characteristics, such as the interests of the subscriber

The characteristics that define a user shape are application-specific and can consist of characteristics other than the preceding ones.

When an SESM web application uses the user-shape mechanisms to customize the web resources for a specific subscriber, the developer performs two sets of development and deployment activities. The developer:

- Organizes web resources into a sparse-tree directory structure
- Implements user-shape decoration based on the web application's requirements

Sparse-Tree Directory Structure

All web resources, such as JSP pages and GIF images, that will be served to the subscriber need to be organized into a sparse-tree directory structure. A *sparse-tree directory structure* is the directory structure of a Cisco SESM web application. Here is a simple example of how a sparse-tree directory structure is organized. If an SESM web application is required to accommodate subscribers whose language is English or Spanish, two sets of GIF images that contain text may be required, one set of images for each language. These two sets of images would be organized into separate English and Spanish directories in the sparse-tree directory structure. For information how you implement this type of directory structure, see the "Using a Sparse-Tree Directory Structure" section on page 3-9.

User-Shape Decoration

Based on application-specific business requirements, the developer needs to determine and implement what the Cisco SESM web application requires for user-shape decoration. An SESM web application includes a group of JSP components that are responsible for *decoration of the user shape*: setting shape *dimensions* (characteristics) such as the device, brand, and locale for a specific subscriber.

Each dimension corresponds to a characteristic of the subscriber (for example, the browser software used by the subscriber). The value of each dimension specifies one or more directories that the SESM software searches—in the sparse-tree directory structure—for requested web resources for this subscriber. The developer determines the dimensions that will be used by the SESM web application and the range of values that will be allowed for each dimension. For information on how you implement user-shape decoration, see the "Decorating a User Shape" section on page 3-16.

Decorators

In an SESM web application, decorators are servlets or JSP pages that implement most of the SESM control and view functionality. A set of decorators is included with the SESM software. An SESM *decorator* modifies some aspect of the current HTTP request, response, session, or application. Most decorators add an attribute to the HTTP request or session. Some decorators modify HTTP response headers or status.

For example, a decorator might add a request attribute specifying a characteristic of the user shape. Another decorator might add a status code to a response header indicating that a resource cannot be found. The SESM controls, such as the `MyAccountControl` and `MyServicesControl`, are decorators that add a JavaBean to the request.

A decorator can be implemented as a Java servlet or as a JSP page.

- The service-provider developer does *not* usually create decorators that are Java servlets. Creating a Java servlet decorator is beyond the scope of this guide.
- The service-provider developer may need to create decorators that are JSP pages. This chapter provides information on creating JSP-page decorators.

Decorator Class

Though the developer does not program decorators that are servlets, it is important to understand the role played by some of the methods that are found in a `Decorator` class. All decorators that are servlets are subclasses of `Decorator`. Three methods in the `Decorator` class are particularly significant:

- `isNecessary`—Tells whether or not decoration is needed. For example, if a `"shape"` attribute is not present in the HTTP session, the `ShapeDecorator` servlet, which is responsible for creating a `Shape` object and adding the attribute, is programmed so that `isNecessary` returns `true`.
- `decorateIfNecessary`—Calls the `decorate` method to perform decoration if `isNecessary` returns `true`.
- `decorate`—Performs decoration (for example, with `ShapeDecorator`, decoration adds the `"shape"` attribute to the HTTP session).

For information on the `Decorator` class, see the Javadoc documentation that is installed with the SESM software.

The SESM software and the NWSP web application provide a complete set of decorators. In many deployments, no additional decorators will be required. However, you can modify or extend the preprogrammed SESM software with JSP-page decorators. The JSP-page decorators created by the service-provider developer are typically dimension decorators or post-decorators.

Dimension Decorators

A *dimension decorator* is a decorator that detects a user-shape characteristic (such as device, brand, or locale) and adds an attribute to the HTTP request that defines the characteristic. Dimension decorators are the SESM mechanism for detecting and setting user-shape characteristics so that the SESM web portal can serve customized web resources to each subscriber.

As an example, consider a dimension decorator that adds an attribute to an HTTP request that specifies the device of the subscriber. If the subscriber is using a PDA, the dimension decorator detects this using HTTP header information. When creating the dimension, this example dimension decorator sets the directory path for the dimension to `"pda"`. The pda directory is where PDA-specific resources reside in directory hierarchy used by the application. The names of the dimension ID and the directory are arbitrary and could be defined differently by the deployer.

The SESM developer can add, modify, or remove dimension decorators based on deployment-specific requirements. For more information on dimension decorators, see the "Decorating a User Shape" section on page 3-16.

Post-Decorators

A *post-decorator* is a decorator that is invoked after a normal servlet or JSP-page decorator executes. A post-decorator is a mechanism for extending the usual, preprogrammed SESM web application functionality. The SESM developer can create custom post-decorators for a variety of application-specific tasks. You configure a post-decorator in the deployment descriptor file (web.xml) so that the post-decorator is invoked after another decorator executes.

For example, the NWSP web application software includes a decorator called `HttpSniffDecorator` to detect HTTP client characteristics such as the browser, device, and operating system that the subscriber is using. If your deployment requires that additional information about the HTTP client be detected, you could create a JSP page to act as a post-decorator.

The sample NWSP web application includes such a post-decorator named httpSniff.jsp, which sends a JavaScript probe to the HTTP client device to try to detect the characteristics of the HTTP client device. It also detects whether the HTTP client device is a WAP phone simulator. When a WAP phone simulator is detected, the NWSP web application uses WML markup language in the content served to the subscriber.

The SESM developer can add, modify, or remove post-decorators based on deployment-specific requirements. For more information on creating post-decorators, see the "Creating JSP-Page Decorators and Post-Decorators" section on page 3-26.

Using a Sparse-Tree Directory Structure

A Cisco SESM web application can use a directory hierarchy that is structured for customizing the SESM web site for each user's shape. The directory structure of the Cisco SESM web application combines two hierarchies:

- Web site pages hierarchy
- User-shape hierarchy

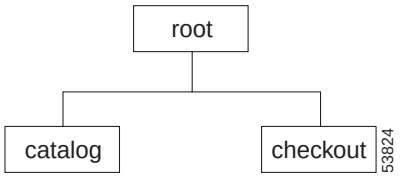
These two hierarchies are developed independently of each other. However, all web resource files are located within one web application because you combine the two hierarchies into one large hierarchy whose root is the web application. For this discussion, a *web resource* might be a JSP page, HTML file, GIF image, or another web application component.

For information on the configuration and programmatic details for implementing the user-shape model, see the "Decorating a User Shape" section on page 3-16.

Web Site Pages Hierarchy

A *web site pages hierarchy* is the directory structure of a conventional web site, which typically includes a single copy of each required resource. For example, consider a web application where the document root contains a page named welcome.html, and subdirectories are named /catalog and /checkout, containing catalog.html and checkout.html respectively. Figure 3-2 shows the web site pages hierarchy.

Figure 3-2 Web Site Pages Directory Hierarchy



User-Shape Hierarchy

The user shapes that need to be accommodated by a Cisco SESM web application determine the application's user-shape hierarchy. A *user-shape hierarchy* is the directory structure of an SESM web site, which may include one or more different instances of each required resource.

User-Shape Example

The following example describes how user shapes help determine the user-shape hierarchy. Assume that the user shapes that need to be accommodated by a Cisco SESM web application have the following characteristics:

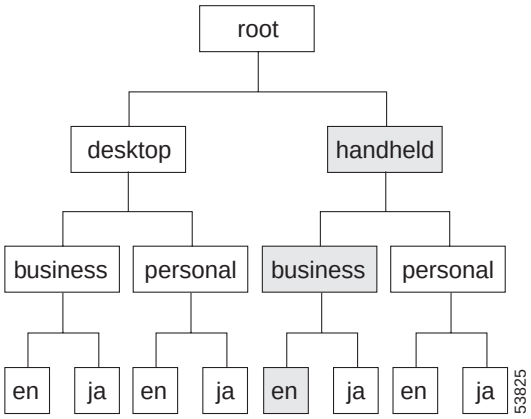
- Devices: desktop and handheld
- Brands: business and personal
- Locales (languages): English (en) and Japanese (ja)

Each of these characteristics (devices, brands, and locales) of the user shape is a *dimension*, and each dimension has one or more *values*. The user shape in this example has three dimensions and two values for each dimension. Each subscriber's user shape has a specific value for each dimension. In this example, the eight possible user shapes are:

- desktop, business, English
- desktop, business, Japanese
- handheld, business, English
- handheld, business, Japanese
- desktop, personal, English
- desktop, personal, Japanese
- handheld, personal, English
- handheld, personal, Japanese

When the directory hierarchy is implemented, the value for each dimension is used for a directory name. In the following example, one or more directories exist named desktop, handheld, business, personal, en, and ja. Figure 3-3 shows the user-shape directory hierarchy.

Figure 3-3 User-Shape Directory Hierarchy



The Cisco SESM web application detects the characteristics of the subscriber and the HTTP client and sets the values of each dimension. For each specific user shape, an SESM web application specifies the directories in the user-shape hierarchy that the Cisco SESM software uses to produce the path for locating a web resource. In Figure 3-3, each of the eight leaves in the directory tree represents one possible user shape. For example, the shaded branch and leaf identifies the user shape that has the values "handheld, business, and English."

Each dimension could be extended. For example, the third dimension (locales) could be extended to include French and Spanish. The number of dimensions is configurable in the web.xml file. The range of values allowed for each dimension is determined by the SESM web application.

Location of Directory Dimensions

You should consider two factors when deciding where a dimension appears within the user-shape directory hierarchy.

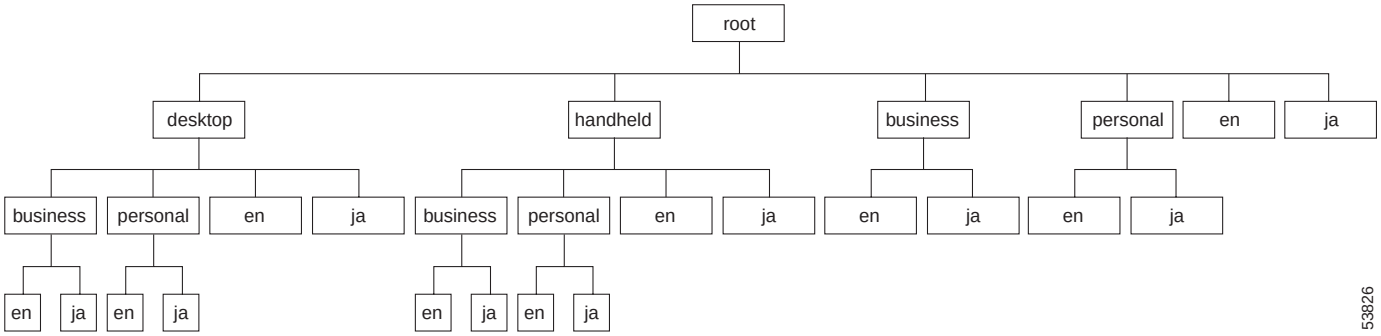
- In general, the more frequently a value appears in the user-shape directory hierarchy, the further down in the hierarchy it is located. As an example, in Figure 3-3 the values of the first dimension (desktop and handheld) appear only once in the hierarchy, but the values of the last dimension (en and ja) appear many times.
- The order of dimensions in the user-shape directory hierarchy affects the order in which the SESM software searches the directories for a web resource. In the web application web.xml file, the `dimensions` initialization parameter for the `ShapeDecorator` servlet lists the dimensions using the same order as the user-shape directory hierarchy. The order given in the `dimensions` parameter determines the order in which the SESM software searches the directories for a web resource.

Except for search order, the dimensions in the hierarchy are independent of each other. For information on the search order, see the "Searches for a Web Resource" section on page 3-13.

Sparse-Tree Directory Structure

The directory hierarchy that a Cisco SESM web application uses is a combination of the web page hierarchy and the user-shape hierarchy. An instance of each resource of the web page hierarchy can reside in every node of the user-shape hierarchy. Figure 3-4 shows the user-shape directory hierarchy expanded to include all possible combinations of dimensions.

Figure 3-4 Fully Expanded User-Shape Directory Hierarchy



An instance of each web resource is not required for each node, but an instance can exist in each node. For example, the resource catalog.html can—but is not likely to—reside in all of the directories shown in Figure 3-4. For example, the catalog.html file, which is located in a /catalog directory, could reside in:

- /catalog/catalog.html
- /handheld/catalog/catalog.html
- /personal/catalog/catalog.html
- /desktop/personal/ja/catalog/catalog.html

The fully expanded hierarchy shown in Figure 3-4 is not likely in a production deployment. The typical deployment makes use of a sparse-tree directory structure because some directories can be omitted. The reasons to omit a directory include:

- If the web resources are identical for all values of a dimension, that dimension can be eliminated. As an example, in Figure 3-3, if the web resources for the devices (desktop and handheld) dimension are identical, that dimension can be omitted. If the web resources for the brands (business and personal) dimension are identical, that dimension can be omitted.
- An empty directory or a directory that contains only empty directories can be pruned from the directory tree. A directory is empty if no user shape will exist for that set of values. For example, if there are no Japanese business users, the /business/ja directory can be omitted.

The web resources that the SESM software finds for a particular user shape can be located in different directories in the directory hierarchy. No one directory in the hierarchy is likely to contain all the resources for a particular user shape.

Implementing the Sparse-Tree Directory

The service provider's web developer might implement the sparse-tree directory in the following manner:

1. Locate the entire web site page hierarchy at the root directory of the user-shape hierarchy. This directory structure will appear as a typical web site.

2. Where required, create a specialized version of a web resource and copy it to the appropriate subdirectory of the user-shape hierarchy.

For example, assume that a logo image for a desktop PC is of high resolution and 32 x 32 pixels, and the logo image for a handheld PC is of a lower resolution and 16 x 16 pixels. The same image is used for business and personal use and by English and Japanese users. The two images, both named /images/logo.gif, are copied into these locations:

- /desktop/images/logo.gif
- /handheld/images/logo.gif

When the web resource /images/logo.gif is requested, the Cisco SESM software uses one of the preceding in the response depending on the value specified for the devices dimension (desktop or handheld) of the subscriber's user shape.

Searches for a Web Resource

When the Cisco SESM web application software searches the sparse-tree directory for a web resource, it uses the algorithm described in this section's examples.

Example 1: Searches for a Web Resource

For Example 1, assume the set of user shapes described in the "User-Shape Example" section on page 3-10. In addition, assume that unique /images/logo.gif files are required for Japanese users—one logo for desktop and one for handheld. The logo.gif resources for Japanese-language users reside in:

- **/desktop/ja**/images/logo.gif
- **/handheld/ja**/images/logo.gif

Two additional logo.gif resources for non-Japanese users reside in:

- **/desktop**/images/logo.gif
- **/handheld**/images/logo.gif

To understand the search algorithm, you need to know these facts about the Cisco SESM web application and one of the initialization parameters of the `ShapeDecorator` servlet.

- The SESM web application sets the values for the dimensions of the user shape, specifying one or more directories for each dimension.
- In the web application web.xml file, the `ShapeDecorator` servlet's `dimensions` initialization parameter determines the order in which the SESM software searches directories for a web resource.

Order in dimensions Initialization Parameter

In the web.xml file, the `dimensions` initialization parameter of the `ShapeDecorator` servlet specifies a set of dimension IDs that the SESM software uses when it creates the dimensions for a subscriber shape. The order in which the dimension IDs are listed is significant because it defines the search order that the SESM software uses to find a web resource for a subscriber.

In the following example, the `dimensions` initialization parameter specifies three directory paths, one path for each dimension in the user shape. The dimensions are for devices, brands, and locales. To simplify this description, the dimensions are designated in the comments as A, B, and C.

```
<servlet>
    <servlet-name>Shape</servlet-name>
    <servlet-class>com.cisco.sesm.navigator.ShapeDecorator</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
       <param-name>dimensions</param-name>
      <param-value>
      device     <!-- dimension A -->
      brand      <!-- dimension B -->
      locale     <!-- dimension C -->
      </param-value>
   ...
```

For a subscriber with the user shape “desktop, business, ja”, assume that the SESM web application specifies that the directory paths associated with the dimensions A, B, and C are as follows:

Dimension	Directory Path
A (devices dimension)	/desktop
B (brands dimension)	/business
C (locales dimension)	/ja

Order of Paths Searched

Given the preceding `dimensions` initialization parameter, user shape, and directory paths, the Cisco SESM software searches the locations shown in Table 3-2 (in the order listed) for /images/logo.gif. As shown in Table 3-2, the search is carried out as follows:

- If the web resource is found at path 1 (**/desktop/business/ja**/images/logo.gif), the Cisco SESM web application uses that resource in its response to the client.
- If the web resource is not found at path 1, the Cisco SESM continues the search at path 2 (**/desktop/business**/images/logo.gif).

The search for /images/logo.gif continues in the order shown in Table 3-2 until the Cisco SESM software finds the resource.

Table 3-2 SESM Search Algorithm: Example 1

Search Order	Directory Path (from document root)	Path Assembly
1	**/desktop/business/ja**/images/logo.gif	A/B/C
2	**/desktop/business**/images/logo.gif	A/B
3	**/desktop/ja**/images/logo.gif	A/C
4	**/desktop**/images/logo.gif	A
5	**/business/ja**/images/logo.gif	B/C
6	**/business**/images/logo.gi	B

Table 3-2 SESM Search Algorithm: Example 1 (continued)

Search Order	Directory Path (from document root)	Path Assembly
7	**/ja**/images/logo.gif	C
8	/images/logo.gif	none (document root)

In the Example 1 search, the Cisco SESM software first finds the web resource /images/logo.gif at /desktop/ja/images/logo.gif and uses that resource in its response to the client. Notice the following about the example:

- The manner in which the directory paths are assembled is determined by the `dimensions` initialization parameter in web.xml. The order of the dimension IDs in the `dimensions` initialization parameter is A, B, and C. The assembly of the directory paths for the search mirrors this order.
- The manner in which directory paths are searched is also determined by the `dimensions` initialization parameter. In the search, the array's last element (C) is the most persistent and is discarded last. The array's first element (A) is the least persistent and is discarded first.

Example 2: Searches for a Web Resource

For this example, assume that the user shapes are as described in the "User-Shape Example" section on page 3-10. However, now assume that an /images/button.gif file resides in the following locations:

- **/desktop/**images/button.gif
- **/business**/images/button.gif
- **/ja**/images/button.gif

Assume that Example 2 uses the same `dimensions` initialization parameter and directory paths as in Example 1. For a user with the shape "desktop, business, ja", the Cisco SESM web application searches the locations shown in Table 3-3, in the order listed, for /images/button.gif.

Table 3-3 SESM Search Algorithm: Example 2

Search Order	Directory Path (from document root)	Path Assembly
1	**/desktop/business/ja**/images/button.gif	A/B/C
2	**/desktop/business**/images/button.gif	A/B
3	**/desktop/ja**/images/button.gif	A/C
4	**/desktop**/images/button.gif	A
5	**/business/ja**/images/button.gif	B/C
6	**/business**/images/button.gi	B
7	**/ja**/images/button.gif	C
8	/images/button.gif	none (document root)

In the Example 2 search, the Cisco SESM software first finds the web resource /images/button.gif at /desktop/images/button.gif and uses that resource in its response to the client. In the search, notice that the last element in the array (C) is the most persistent, and the first element in the array (A) is the least persistent.

Decorating a User Shape

This section describes how a developer configures and customizes a Cisco SESM web application's components for user-shape decoration.

An SESM web application includes a group of JSP components that are responsible for *decoration* of the user shape: setting *dimensions* (characteristics) such as the device, brand, and locale for a specific subscriber.

Each sample SESM web application includes a set of components called *dimension decorators* that are responsible for setting the user-shape characteristics. You can use these SESM-supplied dimension decorators, or create one or more customized dimension decorators to meet application-specific requirements. This section includes information on these topics:

- Configuring User-Shape Dimensions
- Customizing Dimension Decorators

Note Before you read this section on using the decorator components, read the "Using a Sparse-Tree Directory Structure" section on page 3-9. The techniques for user-shape decoration require that the structure of the SESM web site and the techniques for user-shape decoration take a coordinated approach. The explanations in the two sections complement each other.

Configuring User-Shape Dimensions

In a Cisco SESM web application, a `Shape` object encapsulates the set of characteristics that define the web resources available for a specific subscriber. The `Shape` object for a user can include characteristics such as the following:

- Device and browser software used to connect to the web site
- Branding for the web site, such as a brand for business use and a brand for personal use
- Language and country of the user
- Personal characteristics, such as the interests of the subscriber

A `Shape` object consists of one or more dimensions. Each dimension corresponds to a characteristic of the subscriber (for example, the browser software of the subscriber). The value of each dimension specifies one or more directories that the SESM software searches for requested resources for this subscriber. The `ShapeDecorator` servlet creates a `Shape` object for the subscriber.

ShapeDecorator Initialization Parameters

In the web.xml file, the `ShapeDecorator` servlet has two initialization parameters that relate to the dimensions of the user shape:

- `dimensions`
- `postDecorate`

In the following example, the user shape consists of three dimensions: `device`, `brand`, and `locale`.

```
<servlet>
    <servlet-name>Shape</servlet-name>
    <servlet-class>com.cisco.sesm.navigator.ShapeDecorator</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
      <param-name>dimensions</param-name>
      <param-value>
      device
      brand
      locale
      </param-value>
    </init-param>
    <init-param>
       <param-name>postDecorate</param-name>
      <param-value>
      DeviceDimension
      BrandDimension
      LocaleDimension
      </param-value>
    </init-param>
  </servlet>
```

Given the preceding `dimensions` and `postDecorate` initialization parameters, the `ShapeDecorator` servlet, when it is invoked, creates a `Shape` object having three dimensions: `device`, `brand`, and `locale`. Each dimension is initially created with no value.

In the preceding example, three post-decorators are specified in the `postDecorate` parameter of `ShapeDecorator` When the post-decorators execute, these servlets set the values of the three dimensions of the user shape.

The following sections provide more details on the `dimensions` and `postDecorate` initialization parameters.

dimensions Initialization Parameter

In the web.xml file, the `dimensions` initialization parameter specifies a set of zero or more dimension IDs that the SESM software uses when it creates the dimensions for a user shape. The number and order of the dimensions in the `dimensions` parameter can be modified to meet the needs of a specific deployment.

In the `dimensions` parameter, the dimension IDs are strings that are separated by whitespace or commas. The order in which the dimensions are listed is significant because it defines the search order that the SESM software uses to find a resource for a subscriber. For information on search order, see the "Searches for a Web Resource" section on page 3-13.

When you are configuring dimension IDs in the `dimensions` parameter, the dimension ID that is associated with a dimension decorator is determined as follows:

- If the dimension decorator is a JSP page, the dimension ID corresponds to the first argument specified for the `Dimension` constructor when the dimension is created in the JSP page. The following example shows the relevant code from the locationDimension.jsp page.

  ```
  Dimension dim = new Dimension("location", "CityHotel")
  ```

  Given the preceding example, the ID that identifies this dimension decorator is `"location"`.

- If the dimension decorator is a servlet, the dimension ID specified in the `dimensions` parameter is the ID that the servlet returns in its `getId` method. In the typical case, service-provider developers do not create servlet dimension decorators.

Table 3-4 lists the dimension IDs for the SESM-supplied dimension decorators that are part of the NWSP web application.

postDecorate Initialization Parameter

In the web.xml file, the `postDecorate` initialization parameter for `ShapeDecorator` specifies a set of post-decorators that define the values of the dimensions for a user shape. The number of post-decorators corresponds to the number of dimensions in the user shape. The value of each dimension is a directory name or a list of directory names that the dimension's post-decorator determines based on the characteristics of the user shape.

The post-decorators declared with the `postDecorate` parameter give the names of the servlets that set the dimension values for the user shape. The names given in `postDecorate` are the servlet names specified in the web.xml file. Given the preceding sample `postDecorate` servlet initialization, the `DeviceDimension` servlet sets the value of the `device` dimension, the `BrandDimension` servlet sets the value of the `brand` dimension, and so on.

When you declare the post-decorators in the `postDecorate` parameter of `ShapeDecorator`, the names are delimited by whitespace or commas. In the `postDecorate` parameter, the post-decorators can be servlets or JSP pages declared as servlets elsewhere in the web.xml file.

Default Dimension Decorator Values

Like any other decorator, a dimension decorator JSP page or servlet can use a default value for the dimension. The default value comes from the `defaultValue` initialization parameter for the JSP page or servlet as defined in the web application web.xml file.

A dimension decorator servlet or JSP page can retrieve and use the default value for the dimension. The default value is used when the dimension decorator cannot detect a characteristic for that dimension of the subscriber. One advantage to using the `defaultValue` initialization parameter is that the deployer can change the default value without modifying any code.

For example, the locationDimension.jsp of the NWSP web application specifies a default value for the location dimension as follows:

```
<servlet>
    <servlet-name>LocationDimension</servlet-name>
     <jsp-file>/decorators/locationDimension.jsp</jsp-file>
    <load-on-startup>1</load-on-startup>
    <init-param>
         <param-name>defaultValue</param-name>
          <param-value>airport</param-value>
    </init-param>
  </servlet>
```

A JSP page can retrieve the default value using the `getInitParameter` method. In the following example, the locationDimension.jsp page retrieves the default value when creating the location:

```
dim = new Dimension(ID, getServletConfig().getInitParameter("defaultValue"));
```

For an explanation of locationDimension.jsp, see the "Creating or Customizing Dimension Decorators" section on page 3-20.

SESM-supplied Dimension Decorators

Each sample SESM web application includes a set of dimension decorators. You can use these SESM-supplied dimension decorators, or you can create one or more customized dimension decorators to meet application-specific requirements. Table 3-4 lists the SESM-supplied dimension decorators that are part of the NWSP web application.

Note To detect some characteristics of the HTTP client device, the NWSP web application uses a JavaScript facility contained in the javascriptProbe.jsp file. The subscriber's browser must have JavaScript enabled for this facility to work.

Table 3-4 SESM-supplied Dimension Decorators

Dimension Decorator	Dimension ID	Description
brandDimension.jsp	`brand`	Sets the `"brand"` dimension to the value specified in the `"BRAND"` attribute of the SESM session key.
`BrowserDimensionDecorator`	`browser`	Sets the `"browser"` dimension to the name of the browser that is running on the HTTP client device. Possible values are: • an empty string (unknown browser) • `netscape` (Netscape) • `explorer` (Internet Explorer)
`ColorDimensionDecorator`	`color`	Not currently used.
`ConnectionDimensionDecorator`	`connection`	Not currently used.
`DeviceDimensionDecorator`	`device`	Sets the `"device"` dimension to the category of the device running the HTTP client. Possible values are: • `pc` (personal computer) • `pda` (personal digital assistant) • `wap` (WAP phone)
`LocaleDimensionDecorator`	`locale`	Sets the `"locale"` dimension to the value of locale for the current localization (`L10nContext`) context. For example, if the `L10nContext` locale is `"en_GB"`, the value of the dimension is `"en/GB"`. This value specifies two directories (`en` and `GB`) where web application resources are located if the subscriber speaks English and is from Great Britain.
locationDimension.jsp	`location`	Sets the `"location"` dimension to the value specified in the `"LOCATION"` attribute of the SESM session key.

Table 3-4 SESM-supplied Dimension Decorators (continued)

Dimension Decorator	Dimension ID	Description
`MarkupDimensionDecorator`	`markup`	Sets the `"markup"` dimension to the name and version numbers of the markup language that is supported by the browser on the HTTP client device. • an empty string (unknown markup language) • `html` • `html/3` • `html/3/4` • `html/3/layers` • `wml` • `xhtml`
`OSDimensionDecorator`	`os`	Sets the `"os"` dimension to the name of the operating system that is running on the HTTP client device. Possible values are: • an empty string (unknown operating system) • `ce` (Windows CE) • `nt` (Windows NT)
`ScriptDimensionDecorator`	`script`	Sets the `"script"` dimension to the name and version numbers of the script language that is supported by the browser on the HTTP client device. Possible values are: • an empty string (unknown or no script language) • `javascript` • `javascript/1` • `javascript/1/2` • `javascript/1/2/3` • `javascript/1/2/3/4` • `javascript/1/2/3/4/5` • `javascript/2` • `wmlscript`
`SizeDimensionDecorator`	`size`	Not currently used.

If you want to implement the `color`, `connection`, or `size` dimensions, you can obtain the values for these dimensions from the HTTP client browser by using the JavaScript probe facility. You must modify the javascriptProbe.jsp file to set the values for these dimensions.

For more information on the SESM-supplied dimension decorators, see the Javadoc documentation that is installed with the SESM software.

Creating or Customizing Dimension Decorators

For some deployments, the SESM developer may be required to create or customize a dimension decorator. This section provides some guidance on using a JSP page to create or customize a dimension decorator.

Like any other decorator, a decorator that defines a dimension of the user shape can be a JSP page. Using a JSP page to define a dimension has the advantage of not requiring the coding of a Java class or packaging in a JAR file. Each JSP-page dimension decorator performs certain fundamental tasks:

1. Provides a definition for the `jspInit` method and registers itself with the decorator pool
2. Defines a `decorate` method that:
   a. Detects or retrieves a value for some characteristic of the user shape: device, brand, locale, browser, and so on
   b. Based on the detected characteristic, creates and defines a `Dimension` object, setting the value of the dimension to the detected or retrieved value
   c. Updates the user's `Shape` object with the new value of the `Dimension`
3. Invokes the `decorate` method

The following example shows the JSP page locationDimension.jsp that defines the location of the subscriber. The location is the physical location of a mobile client device. For example, the location might be a restaurant, library, or airport. An SESM web application can use the location to serve customized resources to the subscriber.

```
<%@ page import="com.cisco.sesm.logging.Log" %>
<%@ page import="com.cisco.sesm.shape.Shape" %>
<%@ page import="com.cisco.sesm.core.model.SESMSession" %>
<%@ page import="com.cisco.sesm.shape.Dimension" %>
<%@ page import="com.cisco.sesm.navigator.DecoratorPool" %>
<%@ page import="com.cisco.sesm.navigator.ShapeDecorator" %>
<%@ page import="com.cisco.sesm.webapp.decorator.SESMSessionDecorator" %>
<%@ taglib uri="http://www.cisco.com/taglibs/navigator" prefix="nav" %>

<%!     static final String ID = "location"; %>

<%-- (1) Register with the decorator pool --%>
<%! public void jspInit() {DecoratorPool.register(this);} %>

<%-- (2) Define the decorate method --%>
<%!
public void decorate(HttpServletRequest request,
                     HttpServletResponse response)
throws ServletException, IOException
{
     Dimension dim;
   // (3) Get the SESM session.
     SESMSession sesmSession = SESMSessionDecorator.getSESMSession(request, response);

   // (4) Get the SESMSessionKey for this session and get the attribute "LOCATION"
     Object location = sesmSession.getKey().getAttribute("LOCATION");
     Log.debug("SESM attribute 'LOCATION'=", location);

   // (5) Create a dimension for the location using the location object if it is not equal
   //     to null or the defaultValue initialization parameter if location equals null
     if (location != null)
          dim = new Dimension(ID, location.toString());
     else
          dim = new Dimension(ID, getServletConfig().getInitParameter("defaultValue"));

   // (6) Retrieve the Shape object and update its location dimension

     Shape shape = (Shape)request.getSession().getAttribute("shape");
     if (shape != null)
          shape.updateDimension(dim);
     else
          Log.warning("No Shape to decorate with dimension=", dim);
}
%>

<%-- logging --%>
<%! static final String fragment = "/decorators/locationDimension.jsp"; %>
<%@ include file="/logging/enterFragment.jspf" %>

<%-- (7) Exit if invoked by DecoratorPool            -->%
<%-- (which is attempting to force initialization). --%>
<%
if (request.getParameter("DecoratorPool") != null)
     return; //Do nothing. DecoratorPool is forcing initialization.
%>

<%-- (8) Use the decorate tag to execute the decorate method and any post-decorators. --%>
<nav:decorate name="LocationDimension"/>

<%-- logging --%>
<%@ include file="/logging/exitFragment.jspf" %>
```

As shown in the preceding example, when a JSP page is a dimension decorator, it performs the following tasks:

1. Provides a definition for the `jspInit` method, the standard JSP initialization mechanism. The `jspInit` definition registers the JSP page using the `DecoratorPool.register` method. This step is required.
2. Defines the `decorate` method. The definition for the `decorate` method must match the signature of the `Decorator.decorate` method. The signature includes the method name, type, visibility, arguments, and return type. This step is required.

   In this example, the subscriber location is stored in an SESM session attribute.

Note Tasks 3 and 4 describe the method that is used to retrieve an SESM session attribute named `"LOCATION"`, which stores a value for the location dimension. All dimension decorators detect characteristics of the user shape. In this example, the details for retrieving this value of the location dimension are specific to locationDimension.jsp and are not generally applicable to other dimension decorators.

3. Uses the `getSESMSession` method to get the `SESMSession`. `SESMSession` is the main access class for the model. `SESMSession` provides the functionality necessary to manipulate a session and get any data associated with that session. For more information on the `SESMSession` class, see the Javadoc that is installed with the SESM software.
4. Uses `sesmSession.getKey().getAttribute("LOCATION")` to get the `SESMSessionKey` for this session and to retrieve the `SESMSession` attribute named `"LOCATION"`.
5. Creates a new `Dimension` object for the location and initializes the `Dimension` object as follows:
   - The first argument to the `Dimension` constructor is the ID for the dimension. In this example, the ID is the string `"location"`. In the web.xml file, the ID for each dimension decorator is specified in the `dimensions` parameter of the `ShapeDecorator` servlet.
   - The second argument to the `Dimension` constructor is the value (a directory name) for the dimension.

     - If the `SESMSession` attribute named `"LOCATION"` (now stored in `location` variable) is not equal to null, the second argument is the value of that attribute.

     - If the `SESMSession` attribute equals null, the second argument is obtained from the `defaultValue` initialization parameter of the locationDimension.jsp decorator. The initialization parameter is specified in the web application web.xml file.
6. Assigns the value of the `"shape"` session attribute to the `shape` variable and tests whether `shape` is equal to null. The `ShapeDecorator` servlet, which runs when the SESM web application starts, creates a `Shape` object and sets the `"shape"` session attribute to the value of the subscriber's `Shape`. JSP pages use the `"shape"` session attribute to access the `Shape` object.
   - If `shape` is *not* equal to null, the JSP page updates the `"location"` dimension of the `Shape` object with the value (directory name) specified when the `Dimension` was created in Step 5.
   - If `shape` is equal to null, the JSP page logs a warning message.
7. When an unregistered decorator is requested, `DecoratorPool` attempts to force initialization of the decorator in a separate HTTP request. In this case, the JSP page returns without calling the `decorate` method.

   The JSP-page decorator tests whether it is being executed for its intended purpose or to force initialization. The test for forced initialization is the existence of the `DecoratorPool` request parameter. If the `DecoratorPool` parameter exists, the `DecoratorPool` servlet is executing the JSP page to force initialization.

8. Uses the `decorate` tag of the Navigator tag library to invoke the `decorate` method of locationDimension.jsp. The `decorate` tag provides specialized functionality that invokes any pre- and post-decorators that have been declared in the web.xml file for locationDimension.jsp. If a JSP page were to call its `decorate` method directly (rather than using the `decorate` tag), pre-decorators and post-decorators are not invoked.

Modifying Dimension Decorators

The sample Cisco SESM web applications like NWSP contain a set of dimension decorators. Table 3-4 on page 3-19 lists the dimension decorators that are found in NWSP. The service-provider developer can extend the functionality of the SESM-supplied dimension decorators that are servlets, and can modify the functionality of the dimension decorators that are JSP pages.

Servlet Dimension Decorators

Most of the SESM-supplied dimension decorators are implemented as servlets and are not currently modifiable by the service-provider developer. You can use the `postDecorate` initialization parameter to specify a JSP-page post-decorator that extends the behavior of a dimension decorator. You can use the post-decorator mechanism to add to or modify the functionality an SESM-supplied dimension decorator that is implemented as a servlet. For information on creating a post-decorator, see the "Creating JSP-Page Decorators and Post-Decorators" section on page 3-26.

JSP-Page Dimension Decorators

Some of the SESM-supplied dimension decorators are implemented as JSP pages. In the NWSP web application, the dimension decorators that are JSP pages include brandDimension.jsp and locationDimension.jsp. The service-provider developer can directly modify the functionality of a dimension decorator that is a JSP page. As an alternative, you could extend the behavior of a JSP-page dimension decorator by defining a post-decorator.

Adding Dimension Decorators

If an SESM web application requires a new dimension for the user shape, the new dimension decorator is implemented as a JSP-page. When you add a new dimension for the user shape, you must do the following:

1. Create a JSP page for the dimension decorator. See the "Creating or Customizing Dimension Decorators" section on page 3-20.
2. Optionally, create a JSP page for a post-decorator that will be invoked after the dimension decorator. See the "Creating JSP-Page Decorators and Post-Decorators" section on page 3-26.
3. Declare both the dimension decorator and any post-decorators as servlets in the web application's web.xml file.
4. Optionally, in the web.xml declaration of the dimension decorator, use the `postDecorate` initialization parameter to specify the post-decorator that the SESM software invokes after the decorator.
5. In the web.xml declaration of the `ShapeDecorator` servlet, use the `dimensions` and `postDecorate` initialization parameters to configure the new dimension decorator. See the "ShapeDecorator Initialization Parameters" section on page 3-16.

6. Optionally, in the web.xml file, use the `defaultValue` initialization parameter to define a default value for the dimension decorator, the post-decorator, or both. See the "Default Dimension Decorator Values" section on page 3-18.

Modifying SESM Web Application Functionality

The preprogrammed functionality of an SESM web application can be modified in a number of ways including:

- Modifying the Functionality of JSP-Page Views, page 3-25
- Creating JSP-Page Decorators and Post-Decorators, page 3-26
- Using the SESM Deployment Descriptor File, page 3-29

This discussion of functionality changes focuses on the components and web.xml file for the NWSP web application. Before reading this section, become generally familiar with the parts of the NWSP web application by reading the "NWSP User Interface" section on page 2-3.

Modifying the Functionality of JSP-Page Views

The JSP-page views in the sample NWSP web application are preprogrammed with the functionality that most SESM web portals require. Most deployments will use the NWSP pages and possibly make minor changes to the functionality.

Note All changes to the functionality of the JSP-page views in NWSP must be planned and implemented with care. The NWSP controls and views work together in a coordinated manner. Changing one piece of the web application may affect others pieces.

In NWSP, most JSP-page views have a number of functional elements. In some cases, the service-provider developer can modify the functionality of an element. In other cases, modifications are not needed. This section provides some general guidance on whether you can modify a functional element and the types of changes that you might accomplish.

Service List

The service list usually requires no functional modifications. The service list is a tree structure with the services and service groups that the subscriber can select. The service list is dynamically created by the JSP pages based on the service and subscriber information stored in the data repository. For information on the service list, see the "Main Template" section on page 4-10.

Navigation Bar

The navigation bar sometimes requires functional modification. The navigation bar consists of a set of buttons, such as the Services and Accounts buttons, whose display changes based on the actions of the user.

In many deployments, the NWSP navigation-bar functionality requires no changes. Various SESM features (such as service subscription and account management) and the associated navigation-bar buttons require that subscriber have appropriate permissions. The set of buttons that appear in the NWSP navigation bar automatically varies depending on the user's permissions.

In some deployments, the developer may want to add a button to or remove a button from the standard NWSP navigation bar. For example, in some deployments the Help button may not be needed.

- To add a button, you create a new navigation bar. If you want the same functionality as the NWSP navigation bar, you must use Dreamweaver and the Cisco Navigation Bar extension. For information on creating a navigation bar, see Appendix C, "Using the Cisco Navigation Bar Extension."
- To remove a button, you can edit navbar.jsp and delete the lines of HTML that create the button's hyperlink.

Body JSP Pages

The body page of a JSP-page view sometimes requires functional modifications. Each body JSP page contains the functional elements (for example, an HTML form) that appear in the `<body>` section. The functional elements are used by the subscriber to perform a task that is unique to the JSP page. For example, the subscriptionManageBody.jsp page contains the functional elements that the subscriber uses to subscribe to or unsubscribe from services.

In some cases, you can remove functionality from the body JSP page without causing any problems. For instance, you can remove the blocks of HTML code that statusBody.jsp uses to display information about the status of each connected service. As an example, you could remove a field like Elapsed Time from the table of data that statusBody.jsp displays without creating web-application problems or confusing the subscriber.

In cases where a body JSP page uses a form to post data to a NWSP control, you need to determine the effect of removing an input field from the form before making any changes.

- If the input field is not required by the SESM control or model, removing the input field will be harmless. You can make the change.
- If the input field is required by the SESM control or model, removing the input field will not be harmless. You cannot make the change.

Creating JSP-Page Decorators and Post-Decorators

The service-provider developer can create or modify a JSP-page decorator to modify the functionality of an SESM web application. When you implement a decorator as a JSP page, no compiling or packaging is required.

One use for a new JSP-page decorator is as a replacement for an existing servlet decorator. In the web.xml file, you keep the same `<servlet-name>` for the decorator but specify the new JSP-page decorator for the `<servlet-class>`.

A second, more typical use for a new JSP-page decorator is as a post-decorator. A post-decorator is a special type of decorator that is invoked after a normal servlet or JSP-page decorator executes. A post-decorator is a mechanism for extending the usual, preprogrammed SESM web application functionality. In the web.xml file, you use the `postDecorate` initialization parameter in a servlet declaration to specify post-decorators. For information on the `postDecorate` parameter, see the "Using the preDecorate and postDecorate Parameters" section on page B-2.

The `DecoratorByJSP` class makes it possible to implement a decorator with a JSP page. A decorator that is a JSP page is different from a servlet decorator in a number of ways including:

- It has no `decorateIfNecessary` or `isNecessary` methods. In effect, decoration is always needed and always occurs.
- As with any JSP page, it is not a Java class and does not inherit and cannot override any of the methods of the `Navigator` or `Decorator` classes.

For a JSP page to be used as a decorator, the JSP page must do the following:

1. Provide a definition for the `jspInit` method and register itself with the decorator pool.
2. Define a `decorate` method.
3. Use the `decorate` tag from the Navigator tag library to invoke the `decorate` method.

In the NWSP web application, a good example of a JSP-page decorator that a developer might create is httpSniff.jsp. This post-decorator is invoked after `HttpSniffDecorator` and provides additional "browser sniffing" capabilities for detecting characteristics of the HTTP client device that are not found in `HttpSniffDecorator`.

The following NWSP code from httpSniff.jsp shows what is required in a JSP-page decorator. It also shows the categories of tasks a post-decorator might perform.

```
<%@ page import="com.cisco.sesm.logging.Log" %>
<%@ page import="com.cisco.sesm.navigator.HttpSniffBean" %>
<%@ page import="com.cisco.sesm.navigator.DecoratorPool" %>
<%@ page import="com.cisco.sesm.navigator.Navigator" %>

<%@ taglib uri="http://www.cisco.com/taglibs/navigator" prefix="nav" %>
<nav:decorate name="NoCache" />

<%-- (1) Register with the decorator pool --%>
<%! public void jspInit() {DecoratorPool.register(this);} %>

<%-- (2) Define the decorate method --%>
<%!
public void decorate(HttpServletRequest request,
                     HttpServletResponse response)
throws ServletException, IOException
{
<%-- (3) Retrieve and make adjustments to the httpSniffBean --%>
       HttpSniffBean httpSniffBean = (HttpSniffBean)
              request.getSession().getAttribute("httpSniffBean");
       if (httpSniffBean == null)
              throw new ServletException("HttpSniffBean expected.");

       // Sniff the HTTP headers for extra information.
       String userAgent = request.getHeader("User-Agent");
       if ((userAgent != null) &&
              (userAgent.indexOf("OWG1 UP/4.1") >= 0 ||
               userAgent.indexOf("UPG1 UP/4.0") >= 0 ||
               userAgent.indexOf("WinWAP") >= 0))
       {
              httpSniffBean.setClientDeviceName("wap");
       }
<%-- (4) If the client device is JavaScript-enabled, send out a JavascriptProbe --%>

       // Does the client accept JavaScript?
       String accept = request.getHeader("Accept");
       if (accept != null && accept.indexOf("text/html") >= 0)
              httpSniffBean.setClientScriptLanguage("javascript");
       if (accept != null && accept.indexOf("text/javascript") >= 0)
              httpSniffBean.setClientScriptLanguage("javascript");
...
       // Should we send a JavaScript probe?
       String script = httpSniffBean.getClientScriptLanguage();
       if (script != null && script.startsWith("javascript"))
              DecoratorPool.get("JavascriptProbe")
                     .decorateIfNecessary(request, response);
}
%>

<%-- This is where this JSP starts running. --%>
...
<%-- Exit if invoked by DecoratorPool (which is attempting to force initialization). --%>
<% if (request.getParameter("DecoratorPool") != null) return; %>

<%-- (5) Use the decorate tag to execute the decorate method and any post-decorators. --%>
<nav:decorate name="<%=getServletName()%>"/>
```

As shown in the preceding example, when a JSP page acts as a post-decorator, it performs the following tasks:

1. Provides a definition for the `jspInit` method, the standard JSP initialization mechanism. The `jspInit` definition registers the JSP page using the `DecoratorPool.register` method. This step is required.
2. Defines the `decorate` method. The definition for the `decorate` method must match the signature of the `Decorator.decorate` method. The signature includes the method name, type, visibility, arguments, and return type. This step is required.
3. Performs some deployer-specific post-decoration tasks. These tasks typically include:
   - Retrieving a JavaBean that has been created by and had properties set by another decorator (in this example, by `HttpSniffDecorator`)
   - Adjusting one or more properties in the JavaBean based on additional knowledge that the post-decorator has

   In this example, httpSniff.jsp retrieves the bean `HttpSniffBean` and adjusts its `clientDeviceName` when it detects that a WAP phone simulator is the client device.
4. Performs other deployer-specific post-decoration tasks. In httpSniff.jsp, a JavaScript probe is sent to the HTTP client device to detect some of its characteristics. It then makes adjustments to the `httpSniffBean` based on what it is able to detect. The developer could modify httpSniff.jsp to use third-party browser-sniffing software rather than the SESM-supplied JavaScript probe.
5. Uses the `decorate` tag of the Navigator tag library to invoke httpSniff.jsp. The `decorate` tag provides specialized functionality that invokes any pre- and post-decorators that have been declared in the web.xml file for httpSniff.jsp. If a JSP page were to call its `decorate` method directly (rather than using the `decorate` tag), the post-decorators are not invoked.

Using the SESM Deployment Descriptor File

This section provides information on the how you can use the deployment descriptor file (web.xml) to customize the functionality of an SESM web application.

The web.xml file for the NWSP web application, which resides in the *install_dir*\nwsp\docroot\Web-inf directory, provides an example of the elements that you must specify in the deployment descriptor for an SESM web application. You can use this example web.xml file as a template for a file specifically tailored for your deployment.

The web.xml contains the standard J2EE elements found in most such files. For detailed information on the standard elements of a deployment descriptor file, see the *Java Servlet Specification Version 2.3*. The PDF file containing the specification is at:

http://java.sun.com/aboutJava/communityprocess/first/jsr053/servlet23_PFD.pdf

Configuration Information

For all SESM web applications, the general categories of information in a web.xml file are similar. However, the specific elements that are used can differ from one SESM web application to another. This section outlines the general categories that you need to specify in the web.xml file.

In the following examples, the declarations use the elements in the NWSP web application's web.xml file.

In an SESM `web.xml` file, you specify the web application's elements as follows:

1. Declare each logical control as a servlet using the `<servlet>` tag. For example:

```
<servlet>
    <servlet-name>MyAccount</servlet-name>
    <servlet-class>com.cisco.sesm.webapp.control.MyAccountControl</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Note All correctly coded decorators are automatically registered with `DecoratorPool` at servlet initialization. All decorators, including the SESM controls, should be loaded at startup to ensure that they are initialized and registered. The `<load-on-startup>` attribute with a value of 1 loads a servlet when the web server starts.

2. Declare each logical view as a servlet using the `<servlet>` tag. For example:

```
<servlet>
    <servlet-name>MyAccountView</servlet-name>
    <servlet-class>com.cisco.sesm.navigator.VirtualFile</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>vfile</param-name>
    <param-value>/pages/myAccount.jsp</param-value>
    </init-param>
    <init-param>
        <param-name>preDecorate</param-name>
    <param-value>User, NoCache</param-value>
    </init-param>
</servlet>
```

3. Map an appropriate URL to each logical control using the `<servlet_mapping>` tag. For example:

```
<!-- servlet mapping for the control -->

<servlet-mapping>
    <servlet-name>MyAccount</servlet-name>
    <url-pattern>/myAccount</url-pattern>
  </servlet-mapping>
```

4. Declare each decorator (both servlets and JSP pages) and each SESM utility servlet using the `<servlet>` tag. Non-decorator servlets, such as `VirtualFile`, can be loaded on startup for better performance. For example:

```
<servlet>
    <servlet-name>VirtualFile</servlet-name>
    <servlet-class>com.cisco.sesm.navigator.VirtualFile</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

5. If a decorator or SESM utility servlet will be requested as part of a URL, map the appropriate URL to each decorator and utility servlet using the `<servlet-mapping>` tag. For example:

```
<servlet-mapping>
    <servlet-name>VirtualFile</servlet-name>
    <url-pattern>/vfile/*</url-pattern>
  </servlet-mapping>
```

In addition to preceding general configuration guidelines, you may also need to specify deployer-specific configuration information in the web.xml file. The deployer-specific parameters are, for the most part, servlet initialization parameters that the SESM software uses when the servlet is invoked. Appendix B, "SESM Utility Servlets Quick Reference," lists the initialization parameters that the SESM software uses.

When testing an SESM web application, a web.xml file can also include declarations and servlet mappings for test decorators such as `TestDimensionDecorator`. For information on using the test decorators, see the "Using Test Decorators" section on page 2-14.

Configuration Techniques

A Cisco SESM web application's deployment descriptor file (web.xml) provides a number of simple but flexible mechanisms for specifying and invoking a servlet or JSP page. The JSP pages of an SESM web application use some of the same techniques to invoke servlets specified in the URL for a link.

To understand the techniques that are used in the SESM deployment descriptor file, read these sections (in the order listed):

- "Servlet Mapping" section on page 3-31
- "Servlet Chaining" section on page 3-32
- "Mapping a Virtual File Name to an Actual File Name" section on page 3-33
- "preDecorate Initialization Parameter" section on page 3-33

In addition, the "SESM Deployment Descriptor File Techniques Example" section on page 3-34 has an example of how these techniques can be combined to accomplish a set of related web application tasks.

Some of the deployment descriptor file techniques are standard web application mechanisms; other techniques are unique to a Cisco SESM web application. The service-provider developer can combine the techniques in a variety of ways to configure web application functionality and to accomplish web application tasks, such as finding a shape-specific web resource.

Servlet Mapping

Servlet mapping is a standard technique that is used in many deployment descriptor files, including the web.xml file for NWSP. Servlet mapping allows an HTTP request matching a specific URL pattern to be mapped to a particular servlet. The following two examples of servlet mapping are from the NWSP web.xml file:

```
<!-- Example 1: the pattern /cache/* maps to the servlet named Cache -->
  <servlet-mapping>
    <servlet-name>Cache</servlet-name>
    <url-pattern>/cache/*</url-pattern>
  </servlet-mapping>

<!-- Example 2: the pattern /myAccount maps to the servlet named MyAccount -->
<servlet-mapping>
    <servlet-name>MyAccount</servlet-name>
    <url-pattern>/myAccount</url-pattern>
  </servlet-mapping>
```

The preceding two examples show common ways in which servlet mapping is used in the NWSP web.xml file.

- In example 1, the URL pattern `/cache/*` is mapped to the `Cache` servlet name, which is declared elsewhere in web.xml, to be the `CacheDecorator` servlet class. The `CacheDecorator` class is used to tell the HTTP client to cache the SESM response. The URL pattern allows `CacheDecorator` to be invoked in a servlet chain. For example:

  ```
  /cache/pages/accountLogon.jsp
  ```

  In the preceding servlet chain, calling `CacheDecorator` prior to invoking the JSP page given in `/pages/accountLogon.jsp` causes that page to be cached by the client browser. For information on servlet chaining, see the "Servlet Chaining" section on page 3-32.
- In example 2, the URL pattern `/myAccount` is mapped to the `MyAccount` servlet name, which is declared elsewhere in web.xml, to be the `MyAccountControl` servlet class. This use of servlet mapping provides a layer of logical names that can be used in the JSP page navigational links and that are independent of the name of the servlet implementation classes.

Servlet Chaining

Servlet chaining is a standard technique that is used in some deployment descriptor files and in the URLs for links in the JSP pages. For some HTTP requests, the URL can specify that the web application invoke a chain of servlets in a particular order rather than just one servlet. For example:

```
/user/nocache/vfile/pages/home.jsp
```

In the preceding URL, three different servlets are invoked, in order, before the JSP page given in `/pages/home.jsp` is invoked. In the NWSP web.xml file, three servlet names are mapped to these URL patterns:

- `/user/*`
- `/nocache/*`
- `/vfile/*`

With a servlet chain, the HTTP request is sent to the first servlet in the chain. The output from the last servlet in the chain (home.jsp) creates the response sent back to the browser. With an SESM web application, servlets in the chain are usually decorators that modify some aspect of the current HTTP request, response, session, or application. Except for the last servlet in the chain, a decorator in a servlet chain terminates by forwarding the request to the remainder of the URL.

For the NWSP web application, Table 3-5 lists the SESM utility servlets that are sometimes invoked in servlet chains.

Table 3-5 Servlets Invoked in Servlet Chains

Decorator	URL Pattern	Description
`VirtualFile`	`/vfile/*`	Translates a virtual file name into an actual file name according to the current values for the dimensions of the user shape. The actual file name is a URI for a web resource. As part of the translation process, `VirtualFile` attempts to find the resource located by the actual URI.
`CacheDecorator`	`/cache/*`	Tells the HTTP client to cache the HTTP response.
`NoCacheDecorator`	`/nocache/*`	Prevents caching of the HTTP response by the HTTP client.
`UserDecorator`	`/user/*`	Ensures that the User is known (authenticated).

For more information on the SESM utility servlets, see the "SESM Utility Servlets Quick Reference" section on page B-1.

Mapping a Virtual File Name to an Actual File Name

When a Cisco SESM web application employs the user-shape mechanisms for customizing the web resources served to the subscriber, the `VirtualFile` servlet provides important and required functionality. The `VirtualFile` servlet translates a *virtual file name* into an *actual file name* according to the current values for the dimensions of the user shape. The actual file name is a URI for a web resource. `VirtualFile` also attempts to find the resource located by the actual file name and, if it finds the resource, forwards the HTTP request to the resource.

A Cisco SESM web application uses `VirtualFile` when a web resource may differ according to the user shape. For example, if a Cisco SESM web application uses different language-specific icons for a JSP page based on the locale dimension of the user shape, the web application uses `VirtualFile` to find the correct language-specific icon for each subscriber. For an example of how a Cisco SESM web application uses `VirtualFile`, see the "SESM Deployment Descriptor File Techniques Example" section on page 3-34.

In the web.xml file for NWSP, the URL pattern `/vfile/*` is mapped to the `VirtualFile` servlet.

In a servlet chain, `VirtualFile` (`vfile`) must be located immediately before the virtual file name because `VirtualFile` forwards the request to the remainder of the URL. In the following example, the virtual file name for the web resource is `/pages/help.jsp`, and the `/vfile/*` URL pattern is specified immediately before the virtual file name:

```
/user/nocache/vfile/pages/help.jsp
```

preDecorate Initialization Parameter

The `preDecorate` initialization parameter can be specified in the declaration of `VirtualFile` to invoke a sequence of decorator servlets prior to translating a virtual file name into an actual file name. This technique is used in the NWSP web.xml file for the declarations of the logical views to which the SESM controls forward requests. In the following example, the logical view `MyAccountView` is declared using `VirtualFile` and its `preDecorate` parameter:

```
<servlet>
    <servlet-name>MyAccountView</servlet-name>
    <servlet-class>com.cisco.sesm.navigator.VirtualFile</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>vfile</param-name>
        <param-value>/pages/myAccount.jsp</param-value>
    </init-param>
    <init-param>
        <param-name>preDecorate</param-name>
        <param-value>User, NoCache</param-value>
    </init-param>
</servlet>
```

With the preceding declaration, when the SESM control forwards a request to `MyAccountView`, the `VirtualFile` servlet does the following:

1. Invokes, in sequence, the decorator servlets specified in the `preDecorate` parameter: `User` and `NoCache`. Each servlet in the `preDecorate` list is invoked by calling its `decorateIfNecessary` method.
2. Translates the virtual file name (`/pages/myAccount.jsp`) given in the `vfile` parameter into an actual file name (URI) according to the values for the dimensions of the user shape.

Given the servlet mappings in the NWSP web.xml file, using the `preDecorate` parameter in this manner with `VirtualFile` is identical to the following servlet chain:

```
/user/nocache/vfile/pages/myAccount.jsp
```

SESM Deployment Descriptor File Techniques Example

This section uses an example to show how two deployment descriptor file techniques (which were explained in the preceding sections) can be combined to accomplish a set of related Cisco SESM web application tasks:

- Servlet mapping
- Mapping a virtual file name to an actual file name

Using the web.xml file for the NWSP web application, the following example traces the relevant component-to-component flow for this HTTP request:

```
http://someserver:8080/myAccount
```

When the subscriber clicks the My Account button and the web server receives the preceding request, the following occurs if the request is a `GET`:

1. The `MyAccountControl` servlet is invoked because, in web.xml, the URL pattern `/myAccount` maps to the servlet name `MyAccount`. Also, the servlet name declaration for `MyAccount` specifies the `MyAccountControl` servlet class.

```
<!-- Servlet mapping for /myAccount -->
<servlet-mapping>
    <servlet-name>MyAccount</servlet-name>
    <url-pattern>/myAccount</url-pattern>
  </servlet-mapping>

<!-- Servlet name declaration for MyAccount -->
<servlet>
    <servlet-name>MyAccount</servlet-name>
    <servlet-class>com.cisco.sesm.webapp.control.MyAccountControl</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

2. After it finishes processing the request, `MyAccountControl` forwards the request to `MyAccountView`.

```
<servlet>
    <servlet-name>MyAccountView</servlet-name>
    <servlet-class>com.cisco.sesm.navigator.VirtualFile</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>vfile</param-name>
        <param-value>/pages/myAccount.jsp</param-value>
    </init-param>
</servlet>
```

   Notice that, in the web.xml file, the servlet name declaration for `MyAccountView` specifies the `VirtualFile` servlet class and the `vfile` initialization parameter specifies a URI for the view myAccount.jsp.

   If the Cisco SESM web application employs user-shape decoration, `VirtualFile` plays a very important role. As an example of the `VirtualFile` functionality, assume that some content on the myAccount.jsp page can differ depending on the user shape associated with the subscriber.

   When the HTTP request is forwarded to `VirtualFile`, the servlet translates the virtual file name into an actual file name. An actual file name is a URI that has been expanded to include any additional directory paths that are currently associated with the user shape.

For this example, assume the user shape consists of three dimensions (device, brand, and locale) having, respectively, the values `/pda`, `/gold`, and `/fr/CA`. The `VirtualFile` servlet expands the virtual file name `/pages/myAccount.jsp` to the following actual file name:

```
/pda/gold/fr/CA/pages/myAccount.jsp
```

`VirtualFile` then uses the actual URI to search for the resource `/pages/myAccount.jsp` using the search algorithm described in the "Searches for a Web Resource" section on page 3-13.

- If `VirtualFile` finds the resource, it forwards the request to the `/pages/myAccount.jsp` located in the appropriate directory.
- If `VirtualFile` does not find the resource, it throws an exception and displays the following on the client browser:

```
HTTP ERROR: 503 Service Unavailable
javax.servlet.ServletException: Could not find actual file given virtual
file=/pages/myAccount.jsp
RequestURI=/pages/myAccount.jsp
```

If the `/pages/myAccount.jsp` page is found by `VirtualFile`, the web server sends that JSP page's content to the HTTP client. The content is displayed on the subscriber's browser.

Invoking a Decorator

This section provides guidance on where and how a decorator should be invoked.

Decorator Invocation Locations

The manner in which a decorator is used determines the location of the decorator invocation. Many decorators are called as pre-decorators because the processing of the pre-decorator is required before a second decorator can perform its function. Pre-decorators can be invoked from these locations:

- A servlet declaration in the web.xml can define a `preDecorate` parameter.
- The Java code for the SESM control class can call the `addPreDecorator` method. Currently, the service-provider developer does not code control servlets.
- A JSP-page view can use the custom JSP tag `<nav:decorate name="servletName"/>`.

All three invocation locations are equivalent. Calling a decorator more than once is slightly inefficient but harmless.

The SESM software follows these guidelines to determine where to invoke a decorator:

- If a SESM control knows that the decorator must run, the control adds the decorator to its list of pre-decorators using the `addPreDecorator` method. This location is required when the control would fail if the decoration does not take place.
- If a control or view does not require a decorator to run, the decorator should be specified in the `preDecorate` parameter. This parameter is specified in the `<servlet>` declaration of the control or view where the deployer can easily change the configuration.

For example, in NWSP, the `StatusControl` and the Status view JSP pages might be capable of displaying the Status page with or without an authenticated user. If the deployment allows an unauthenticated user to display the Status page, the control and view would *not* invoke the `UserDecorator` servlet, which ensures that the subscriber is authenticated. However, if the deployment allows only an authenticated user to display the Status page, `UserDecorator` should be invoked through the `preDecorate` parameter in the `<servlet>` declaration for the view.

Decorator Invocation Methods

An SESM web application can invoke a decorator in a number of different ways. In general, invoking a decorator as a servlet is more expensive than other invocation methods. An SESM decorator can be configured so that, when it is called, it uses pre-decorators and post-decorators to extend or modify the functionality. For the examples that follow, the web.xml file entry for the `UserDecorator` servlet is as follows:

```
<servlet>
    <servlet-name>User</servlet-name>
    <servlet-class>
    com.cisco.sesm.webapp.decorator.UserDecorator
    </servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>preDecorate</param-name>
        <param-value>Decorator1, Decorator2</param-value>
    </init-param>
    <init-param>
        <param-name>postDecorate</param-name>
        <param-value>Decorator3, Decorator4</param-value>
    </init-param>
</servlet>
```

When a JSP page invokes a decorator, such as `UserDecorator`, in one of the following ways, any decorators in the `preDecorate` and `postDecorate` lists are called if the decorator being declared in `<servlet-class>` (in this example, `UserDecorator`) requires decoration.

- Invoking the decorator as a servlet (for example, through an HTTP request, or by forwarding to the decorator or including it into a JSP page).
- Using the `decorate` tag of the Navigator tag library. For example:

  ```
  <nav:decorate name="User" />
  ```

- Directly invoking the `decorateIfNecessary` method. For example:

  ```
  DecoratorPool.get("User").decorateIfNecessary(request, response);
  ```

In all cases, the pre-decorators and post-decorators are invoked by calling the `decorateIfNecessary` method of each.

Note When a decorator is invoked directly by calling its `decorate` method, the decorators in the `preDecorate` and `postDecorate` lists are not called.

For information on the `preDecorate` and `postDecorate` initialization parameters, see the "Using the preDecorate and postDecorate Parameters" section on page B-2.