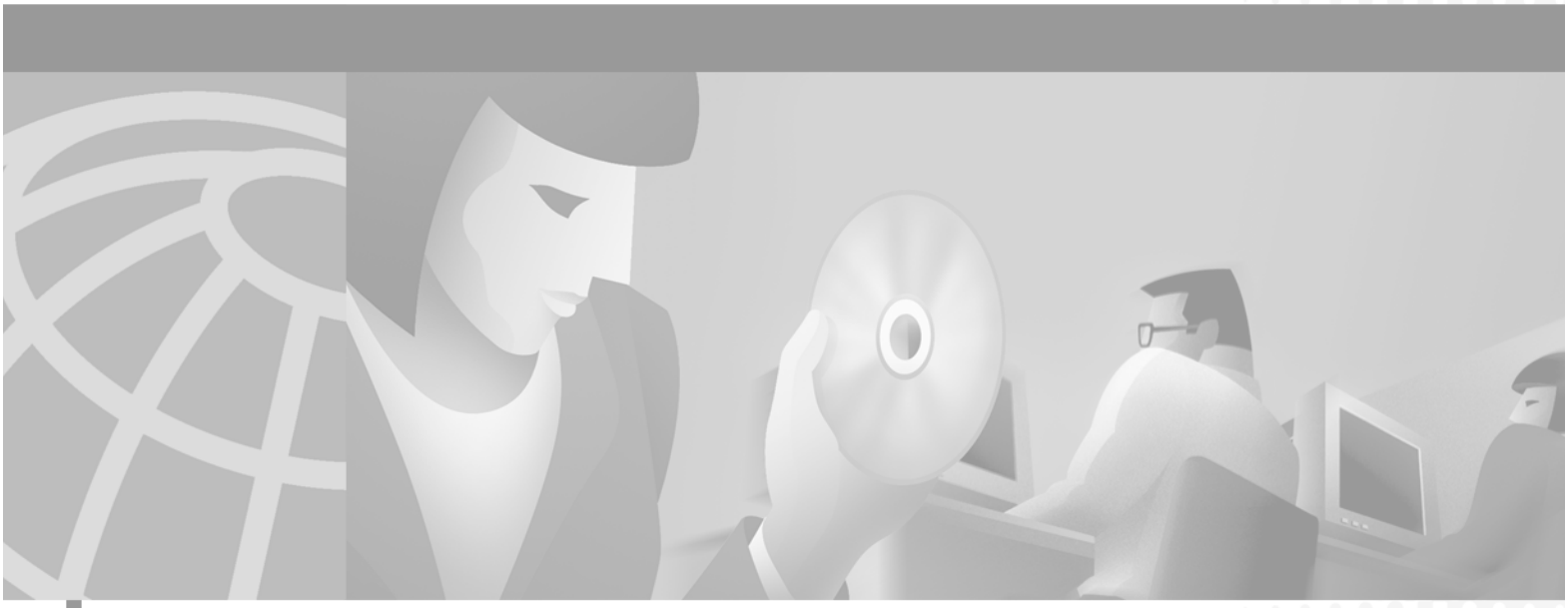


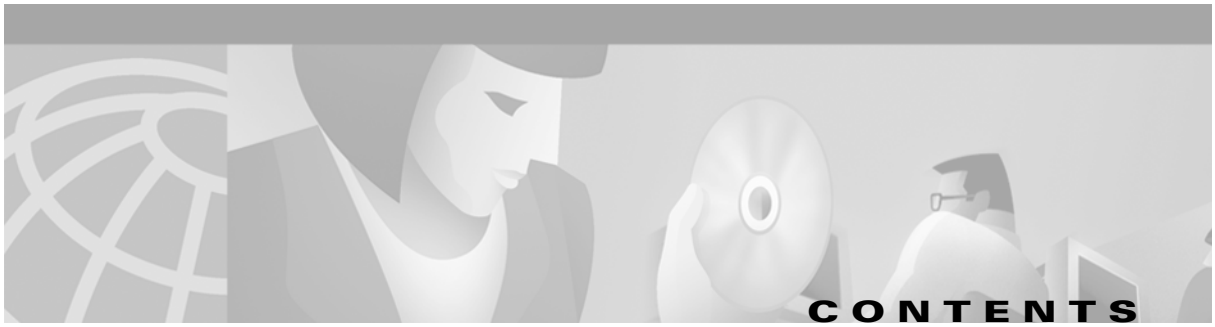
Cisco Subscriber Edge Services Manager Web Developer Guide

SESM Release 3.1(3)
March 2002

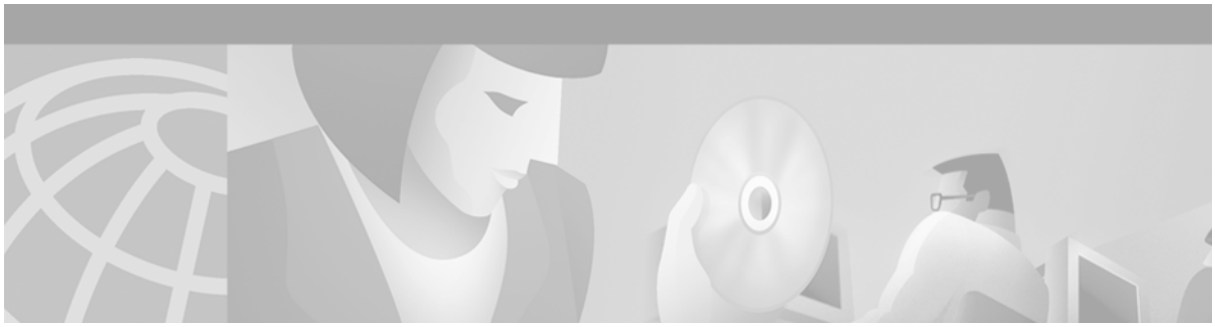
Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

Customer Order Number: DOC-OL2068=
Text Part Number: OL-2068-01

Cisco Subscriber Edge Services Manager Web Developer Guide



CONTENTS



About This Guide

This preface has information about the *Cisco Subscriber Edge Services Manager Web Developer Guide* and contains the following sections:

Document Objectives

This guide explains how to develop a Cisco Subscriber Edge Services Manager (Cisco SESM) web application. It describes SESM web application components and techniques.

Audience

This guide is intended for web designers and web developers responsible for the look-and-feel, branding, and functionality of an SESM web application. It is intended for web designers who will use the HTML design and integrate it with the SESM web components using JavaServer Pages.

Document Organization

This guide includes the following chapters and appendixes:

Chapter	Title	Description
Chapter 1	SESM Web Development Overview	Provides an overview of a Cisco SESM system and an SESM web application.
Chapter 2	Basic SESM Customization and Development	Explains how to do basic customization of the look-and-feel elements of an SESM web application. Provides information on web application development.
Chapter 3	Advanced SESM Customization	Explains some of the advanced customization techniques that you can use with a Cisco SESM web application.
Chapter 4	Sample SESM Web Applications	Provides information on the sample SESM web applications and solutions and describes how a developer can use and modify the sample web components.
Chapter 5	SESM Internationalization and Localization	Explains the SESM components and techniques that help a deployer internationalize and localize an SESM web application.
Appendix A	SESM Tag Libraries	Provides information on configuring an SESM tag library and describes the SESM Iterator, Navigator, and Shape tag libraries.
Appendix B	SESM Utility Servlets Quick Reference	Provides quick-reference information on the utility Java servlets that are used in an SESM web application.
Appendix C	Using the Cisco Navigation Bar Extension	Explains how you install and use the Cisco Navigation Bar extension to Dreamweaver.
Index		

Document Conventions

The following conventions are used in this guide:

- **Boldface** font is used for user action, commands, and keywords.
- *Italic* font is used for emphasis, new terms, and elements such as a file name for which you supply a value.
- `Computer` font is used for code that appears on a JavaServer Page.

Note Means *reader take note*. Notes contain helpful suggestions or references to materials not covered in the manual.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Related Documentation

The following documents are relevant to Cisco SESM software and web development:

- *Release Notes for the Cisco Subscriber Edge Services Manager Release 3.1(3)*
- *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide*
- *Cisco Distributed Administration Tool Guide*
- *Service Selection Gateway*

On Cisco Connection Online at www.cisco.com, you can find information on each of the SSG Features in IOS Release 12.2(4)B.

Obtaining Documentation

The following sections explain how to obtain documentation from Cisco Systems.

World Wide Web

You can access the most current Cisco documentation on the World Wide Web at the following URL:

http://www.cisco.com

Translated documentation is available at the following URL:

http://www.cisco.com/public/countries_languages.shtml

Documentation CD-ROM

Cisco documentation and additional literature are available in a Cisco Documentation CD-ROM package, which is shipped with your product. The Documentation CD-ROM is updated monthly and may be more current than printed documentation. The CD-ROM package is available as a single unit or through an annual subscription.

Ordering Documentation

Cisco documentation is available in the following ways:

- Registered Cisco Direct Customers can order Cisco product documentation from the Networking Products MarketPlace:

  http://www.cisco.com/cgi-bin/order/order_root.pl

- Registered Cisco.com users can order the Documentation CD-ROM through the online Subscription Store:

  http://www.cisco.com/go/subscription

- Nonregistered Cisco.com users can order documentation through a local account representative by calling Cisco corporate headquarters (California, USA) at 408 526-7208 or, elsewhere in North America, by calling 800 553-NETS (6387).

Documentation Feedback

If you are reading Cisco product documentation on Cisco.com, you can submit technical comments electronically. Click **Leave Feedback** at the bottom of the Cisco Documentation home page. After you complete the form, print it out and fax it to Cisco at 408 527-0730.

You can e-mail your comments to bug-doc@cisco.com.

To submit your comments by mail, use the response card behind the front cover of your document, or write to the following address:

Cisco Systems
Attn: Document Resource Connection
170 West Tasman Drive
San Jose, CA 95134-9883

We appreciate your comments.

Obtaining Technical Assistance

Cisco provides Cisco.com as a starting point for all technical assistance. Customers and partners can obtain documentation, troubleshooting tips, and sample configurations from online tools by using the Cisco Technical Assistance Center (TAC) Web Site. Cisco.com registered users have complete access to the technical support resources on the Cisco TAC Web Site.

Cisco.com

Cisco.com is the foundation of a suite of interactive, networked services that provides immediate, open access to Cisco information, networking solutions, services, programs, and resources at any time, from anywhere in the world.

Cisco.com is a highly integrated Internet application and a powerful, easy-to-use tool that provides a broad range of features and services to help you to

- Streamline business processes and improve productivity
- Resolve technical issues with online support
- Download and test software packages
- Order Cisco learning materials and merchandise
- Register for online skill assessment, training, and certification programs

You can self-register on Cisco.com to obtain customized information and service. To access Cisco.com, go to the following URL:

http://www.cisco.com

Technical Assistance Center

The Cisco TAC is available to all customers who need technical assistance with a Cisco product, technology, or solution. Two types of support are available through the Cisco TAC: the Cisco TAC Web Site and the Cisco TAC Escalation Center.

Inquiries to Cisco TAC are categorized according to the urgency of the issue:

- Priority level 4 (P4)—You need information or assistance concerning Cisco product capabilities, product installation, or basic product configuration.
- Priority level 3 (P3)—Your network performance is degraded. Network functionality is noticeably impaired, but most business operations continue.
- Priority level 2 (P2)—Your production network is severely degraded, affecting significant aspects of business operations. No workaround is available.
- Priority level 1 (P1)—Your production network is down, and a critical impact to business operations will occur if service is not restored quickly. No workaround is available.

Which Cisco TAC resource you choose is based on the priority of the problem and the conditions of service contracts, when applicable.

Cisco TAC Web Site

The Cisco TAC Web Site allows you to resolve P3 and P4 issues yourself, saving both cost and time. The site provides around-the-clock access to online tools, knowledge bases, and software. To access the Cisco TAC Web Site, go to the following URL:

http://www.cisco.com/tac

All customers, partners, and resellers who have a valid Cisco services contract have complete access to the technical support resources on the Cisco TAC Web Site. The Cisco TAC Web Site requires a Cisco.com login ID and password. If you have a valid service contract but do not have a login ID or password, go to the following URL to register:

http://www.cisco.com/register/

If you cannot resolve your technical issues by using the Cisco TAC Web Site, and you are a Cisco.com registered user, you can open a case online by using the TAC Case Open tool at the following URL:

http://www.cisco.com/tac/caseopen

If you have Internet access, it is recommended that you open P3 and P4 cases through the Cisco TAC Web Site.

Cisco TAC Escalation Center

The Cisco TAC Escalation Center addresses issues that are classified as priority level 1 or priority level 2; these classifications are assigned when severe network degradation significantly impacts business operations. When you contact the TAC Escalation Center with a P1 or P2 problem, a Cisco TAC engineer will automatically open a case.

To obtain a directory of toll-free Cisco TAC telephone numbers for your country, go to the following URL:

http://www.cisco.com/warp/public/687/Directory/DirTAC.shtml

Before calling, please check with your network operations center to determine the level of Cisco support services to which your company is entitled; for example, SMARTnet, SMARTnet Onsite, or Network Supported Accounts (NSA). In addition, please have available your service agreement number and your product serial number.

CHAPTER 1

SESM Web Development Overview

This chapter provides an overview of a Cisco Subscriber Edge Services Manager (Cisco SESM) system and a Cisco SESM web application. It also provides a high-level look at the web components and techniques used to develop a Cisco SESM web application.

Cisco SESM System

Cisco SESM is an extensible set of applications for providing on-demand services, service management, and subscriber management to the broadband, wireless LAN, and mobile wireless markets. Cisco SESM is part of a Cisco solution that allows subscribers of DSL, cable, wireless, and dial-up to simultaneously access multiple services provided by different Internet service providers, application service providers, and Corporate Access Servers.

Cisco SESM allows a service provider to create a customized web application that provides a branded web portal for individual subscribers. Through the Cisco SESM web portals, subscribers can have simultaneous access to the Internet, corporate intranets, gaming, and other entertainment-based services. After logging on and being authenticated to the system, subscribers access their own personalized services by pointing and clicking.

Depending on the SESM software that is used, a deployed SESM web application can be configured for one of two modes:

- *RADIUS mode*—Service and subscriber information is stored in a RADIUS server.
- *LDAP mode*—Service, subscriber, and policy information is stored in an LDAP-compliant directory, which is accessed with the Directory Enabled Service Selection (DESS) application programming interfaces of Cisco Subscriber Policy Engine (SPE).

A Cisco SESM web application allows subscribers to:

- Select or deselect services to which they are subscribed
- Subscribe to or unsubscribe from services they are authorized to access (DESS mode only)
- Change account details, such as address information and passwords (DESS mode only)
- Create subaccounts for other family members (DESS mode only)
- View the status of service connections
- View system messages

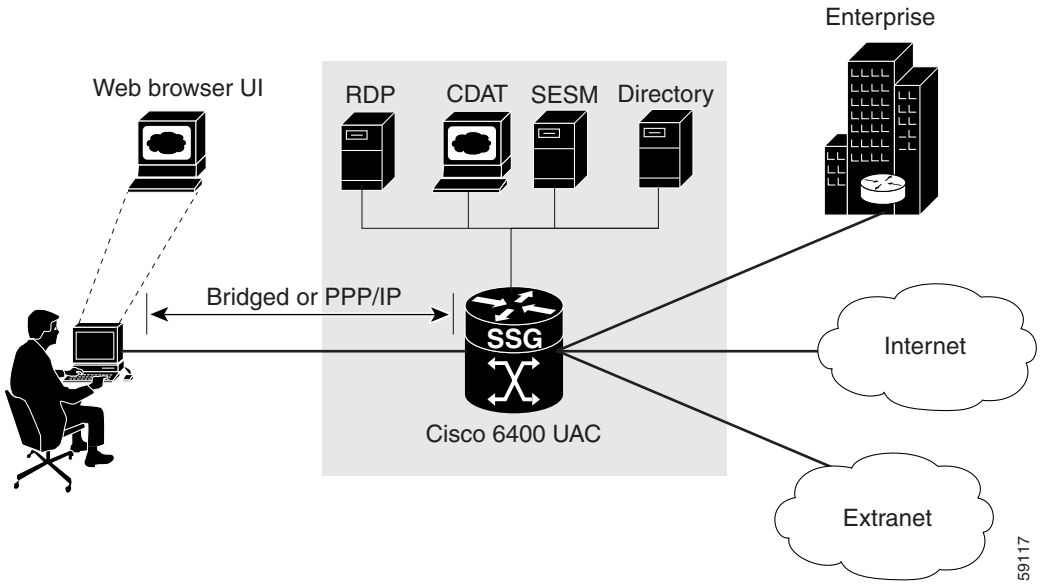
A Cisco SESM system includes one or more SESM servers running one or more Java 2 Enterprise Edition (J2EE) web servers.

The Cisco SESM system components (Figure 1-1) can include the Cisco Service Selection Gateway (SSG), a Cisco IOS software feature module that is available on the Cisco 6400, 7200, and 7400 platforms. Cisco SSG works in conjunction with SESM to provide a number of features including the prepaid billing feature. The HTTP Redirect feature of the SSG works with an SESM Captive Portal web application.

Figure 1-1 shows an SESM configuration that includes the components for an LDAP-compliant directory implementation.

- RDP (RADIUS-DESS Proxy) is a RADIUS proxy server that uses the DESS class libraries to translate RADIUS into Lightweight Directory Access Protocol (LDAP) so that service and subscriber information in an LDAP-compliant directory can be accessed.
- CDAT (Cisco Distributed Administration Tool) is a web application for creating and maintaining service and subscriber information in an LDAP-compliant directory.

Figure 1-1 SESM System Components



For detailed information on SESM system components, see the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide*.

Cisco SESM Web Applications

A Cisco SESM *web application* is a collection of associated web resources that can include JavaServer Pages (JSP), servlets, utility classes, static documents (such as HTML or WML files), images, and other data. An SESM web application includes:

- A business model of the relationship between subscribers and services. In DESS mode, the business model supports service selection, service subscription, subscriber self-care, and service and subscriber management.
- Components that allow the SESM business model to communicate with other system components such as an SSG, a RADIUS AAA server, and an LDAP-compliant directory.
- Components for internationalization and localization of an SESM web application.

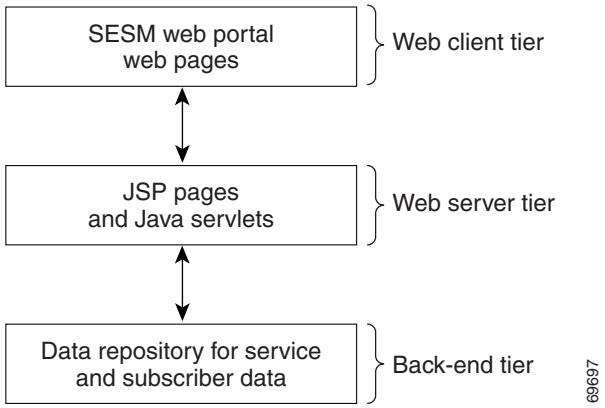
- Development guidelines that allow an SESM web application to be deployed and configured in the context of a J2EE web server.

The Jetty web server from Mortbay.com is installed with the SESM software—though a Cisco SESM web application can be deployed with any J2EE-compliant web server. From a development perspective, a Cisco SESM web application has three tiers:

- Web client tier—The subscriber accesses the SESM user interface through a web browser.
- J2EE web server tier—A set of JSP pages and Java servlets receives requests from the web client, processes the requests by communicating with back-end components, and renders and sends a reply to the client.
- Back-end tier—One or more data repositories (such as a RADIUS authentication, authorization, and accounting (AAA) server or an LDAP directory) store service and subscriber information.

Figure 1-2 illustrates this three-tiered perspective.

Figure 1-2 Three-tiered SESM Web Application



The interactions between an SESM web application and the back-end components are implemented with a set of specialized SESM programming interfaces. Many of these interactions are preprogrammed in the JSP pages and are configurable by the SESM deployer. The functionality of the preprogrammed JSP pages can be modified or extended by the service-provider's developer. Because no extensive Java programming is required, SESM web development can be completed in a much shorter period of time.

For the service provider, SESM web development involves two distinct roles:

- Web designer—Responsible for the HTML or WML design: the look, branding, and organization of the web site.
- Web developer—Responsible for integrating the HTML or WML design with the SESM web components using JSP pages.

The SESM software provides components and techniques for the customization and localization of the web pages. The web designer and web developer determine what customization is needed and use the SESM components and techniques to modify the JSP pages and other SESM components in the required manner.

Sample Cisco SESM Web Applications

A complete sample Cisco SESM web application is included with the SESM software: New World Service Provider (NWSP). The NWSP web application gives the web developer a fully functional set of SESM web components that demonstrate much of what can be done in an SESM web application. The NWSP sample web application can also be used as the basis for creating an SESM web application that meets the service provider's branding, look-and-feel, and functional requirements.

Two other sample web applications demonstrate how SESM might be used for service selection on specific devices:

- The PDA (personal digital assistant) web application is designed for handheld devices.
- The WAP (Wireless Application Protocol) web application is designed for mobile phones.

Conventional Web Sites: A Problem

Most conventional web sites on the World Wide Web are structured in a manner that works well as long as certain assumptions about the user are met. For example, the user:

- Has a PC running Microsoft Windows
- Has a browser that is a recent version of Microsoft Internet Explorer or Netscape Navigator
- Understands English (with American spelling)

In the past, these assumptions were true for most users of conventional web sites.

The problem with conventional web sites is that they cannot adapt their content and format to the variety of user characteristics that are currently on the web. Today, web-based interfaces must accommodate more variety on the World Wide Web in the form of:

- Diversity of devices including PCs, handheld computing devices, and smart phones
- Diversity of bandwidths and software technologies such as HTTP and WAP
- Diversity of markup languages including HTML, HDML, XML, and WML
- Diversity of languages and cultures

Sometimes it is required that the same information or services be provided to the same subscriber with a variety of applications, devices, and locales.

Customer circumstances such as the device, bandwidth, and language are outside of the control of the web site. Other customer characteristics are imposed by the web site itself. Is this a new or existing customer? With which brands is the web site associated? What level of service has the customer selected?

SESM Technology: A Solution

With SESM, the set of characteristics (for example, device, brand, and locale) that define a subscriber is called the user's *shape*. The web components, directory hierarchy, and infrastructure of a Cisco SESM web application are designed to provide an SESM web portal that is customized for each user's shape.

Because an SESM web application is designed to detect and adapt to each user's shape, the Cisco SESM web portal can provide customer-tailored content and service offerings as well as a high degree of brand identity. As an example, an SESM web application can identify the location of the subscriber and provide location-specific pages and service offerings.

How does the Cisco SESM dynamically adapt to the user's shape? There are a number of possible strategies for providing customized content:

- Strategy 1—Make the existing web pages adapt for different types of users.
- Strategy 2—Create a separate set of web pages for each type of user.
- Strategy 3—A combination of strategy 1 and strategy 2.

Strategy 1

Strategy 1 can make web pages complex, error prone, and difficult to maintain. For example, the presentation of a web page targeted for a Personal Digital Assistant (PDA) is different from the presentation of the same page targeted for a PC. These size differences often require dynamic HTML scripting to accommodate the differences in page content and layout. Implementing and maintaining these pages can be difficult.

Strategy 2

Strategy 2 is often used when there is a requirement to support multiple languages. The entire web site is copied, and one language is replaced with another. For example, an English-language web site might be copied, and the English-language elements including text, currency symbols, formats of dates, and formats of numbers are replaced with Japanese-language elements. The end result is one web site for each language that is supported.

Strategy 2 ignores the commonality of elements on the multiple sets of web pages. As an example, the company logo might be the same on the pages in each web site but would need to be included in the set of resources in each web set. When you change the logo on one web site, the change has to be replicated on the other web sites. This approach is time-consuming and error prone.

If you need to support English, Japanese, and Spanish, and desktop PCs, color PDAs, and monochrome PDAs, both strategy 1 and strategy 2 become more complicated. Strategy 1 requires complex dynamic HTML techniques, and strategy 2 requires nine separate web sites.

Strategy 3

Strategy 3 is the approach to user customization that a Cisco SESM web application takes. This approach combines the most useful aspects of strategy 1 and strategy 2 while it attempts to minimize the drawbacks of both techniques. Cisco SESM software provides easy-to-use mechanisms for customizing web content and format based on each user's shape.

- SESM web components—The SESM web components are designed so that they can be easily changed to meet the service provider's branding and look-and-feel requirements. The SESM web components include:
  - Easy-to-customize JSP pages
  - Templates that allow an entire set of web pages to be updated when a template is changed
  - Images and icons that are shipped with the native image-editor source files so they can be quickly modified
- SESM directory hierarchy—The SESM web application uses a *sparse-tree directory structure*. With this directory structure, though some web site resources may exist in more than one location (as in strategy 2), the need for multiple copies of the same resource is greatly reduced.
- SESM web application infrastructure—The SESM web application includes Java servlets, JSP pages, and specially designed Java classes that allow the SESM web site to be customized and localized for each subscriber.

For detailed information on each of the preceding elements of the SESM web application, see Chapter 2, "Basic SESM Customization and Development" and Chapter 3, "Advanced SESM Customization."

Hardware and Software Requirements for Development

On the development machine, the following software must be installed and set up correctly:

- Cisco SESM Release 3.1(3)
- Java 2 SDK, Standard Edition, Version 1.2.2 or later, which is available for downloading at:

  http://java.sun.com/products/jdk/1.2/index.html

In addition to the preceding software, the other hardware and software requirements for SESM web development include the same requirements that apply to deploying an SESM web application except that no separate Java Runtime Environment (JRE) is needed because the Java 2 SDK is installed. For information on these other requirements and how to install and configure the SESM software, see the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide*.

The development machine must have a J2EE-compliant web server installed and set up correctly. The Jetty web server is included with the SESM software. For information on installing and configuring a web server, refer to the instructions that apply to your web server.

Demo Mode and Development

You can install or configure the SESM software for demonstration mode to observe how the NWSP, PDA, or WAP web applications work. Demonstration mode is also useful during some phases of web-application development because this mode does not require other system components such as a configured Cisco edge router and SSG. For more information on demonstration mode, see the "Demo Mode" section on page 2-10.

Environment Variables

The installation instructions for the Java 2 SDK describe how to set the required environment variables. For information on where to find the installation instructions, see the README file that is installed with the Java 2 SDK. Before you start Cisco SESM web development, check that the environment variables shown in Table 1-1 are set to the indicated locations.

Table 1-1 Java 2 SDK Environment Variables

Environment Variable	Value
`JDK_HOME`	The location of the Java 2 SDK Standard Edition installation if that SDK is used.
`PATH`	The location of the `bin` directory of the Java 2 SDK installation (for example, C:\jdk1.2.2\bin). This allows you to run the SDK executables from any directory.

If you use the `CLASSPATH` environment variable to tell the Java compiler where to find SESM-related class files, you need to set `CLASSPATH` to the location of these files. For information on the class libraries needed for compiling an SESM web application, see the "SESM Class Libraries" section on page 2-7.

Recommended Development Tools

The Cisco SESM software includes web components that were developed with Dreamweaver UltraDev 4, a visual editor for creating and managing web sites and pages, and Fireworks 4, a web graphics design and production facility. We recommend these two state-of-the-art tools for SESM web development because with them you can create a customized SESM web application more quickly and easily.

Neither Dreamweaver UltraDev nor Fireworks is required to develop an SESM web application. However, you can more easily modify some SESM web components using these tools. SESM web components that require Dreamweaver UltraDev and Fireworks include Dreamweaver templates for creating and modifying page layout and appearance.

Dreamweaver UltraDev4 has a Live Data window feature that shows actual dynamic content that is generated by JSP pages. When you are developing a Cisco SESM web application, the Live Data allows you to make changes to the JSP pages in a live-data environment.

Many SESM images and icons are provided in Portable Networks Graphics (PNG) format so they can be easily customized in Fireworks. Image editors other than Fireworks may be limited in their ability to edit these PNG-formatted images and icons.

For information on Dreamweaver UltraDev 4 and Fireworks 4, see the Macromedia web site at:

http://www.macromedia.com

Both Dreamweaver UltraDev 4 and Fireworks 4 are available at the web site for a free 30-day evaluation.

Learning about SESM Web Application Development

You should become familiar with the SESM components and techniques by reading this document and experimenting with a sample SESM application such as NWSP. Table 1-2 lists some topics with which the developer should be familiar.

Table 1-2 SESM Web Development Reading Path

For information on this topic	Read this chapter or section
Overview of SESM components and techniques	Chapter 1, "SESM Web Development Overview"
Customizing an SESM web application's look and feel	"Changing the Look-and-Feel Elements" section on page 2-2
Using the web components in a sample SESM web application	"Using a Sample SESM Web Application" section on page 2-3
Developing an SESM web application, including the steps needed to compile JSP pages	"Developing an SESM Web Application" section on page 2-6
Serving SESM web pages that match the user's shape	"User Shapes and User-Shape Decoration" section on page 3-7 and "Decorating a User Shape" section on page 3-16
Using advanced customization techniques, such as modifying SESM web-application functionality	Chapter 3, "Advanced SESM Customization"

Table 1-2 SESM Web Development Reading Path (continued)

For information on this topic	Read this chapter or section
Learning about the sample SESM web applications	Chapter 4, "Sample SESM Web Applications"
Internationalizing and localizing an SESM web application	Chapter 5, "SESM Internationalization and Localization"

Helpful Web Sites and Other Resources

For information on J2EE, web application development, JSP pages, and other topics, the web provides many resources including the Java Developer Connection at http://developer.java.sun.com/developer. Table 1-3 lists some other recommended web-development resources.

Table 1-3 Web Development Resources

Resource	Description
Core Servlets and JavaServer Pages (by Marty Hall, Sun Microsystems Press, 2000)	Excellent book for learning about JSP pages.
Java Servlet Specification Version 2.3 (Sun Microsystems, Inc., 2000)	Useful general information on web applications, deployment descriptor files, and other related topics. The PDF file containing the specification is at available at http://java.sun.com.
Java servlet or JSP class library documentation	Available as Javadoc in the /doc directory at the Java 2 SDK installation location.
HTML and Cascading Style Sheet specifications	Useful reference material is available at the World Wide Web Consortium (W3C) web site located at http://www.w3.org.
Beginning WAP, WML, and WMLScript (by Wei Meng Lee, Soo Mee Foo, et al, Wrox Press Ltd., 2000)	Very good resource for learning about WAP and WML.

If you plan to use Dreamweaver UltraDev or Fireworks as development tools and are not familiar with their use, those facilities have their own documentation, Help systems, and web resources. For a list of documents available on the web, go to the Dreamweaver UltraDev and Fireworks support centers at:

http://www.macromedia.com/support/ultradev/documentation.html

and

http://www.macromedia.com/support/fireworks/documentation.html

CHAPTER 2

Basic SESM Customization and Development

In some Cisco SESM deployments, you can use one of the sample SESM web applications as a starting point and accomplish look-and-feel modifications to the JSP pages without changing the preprogrammed SESM software. This chapter explains how to do basic customization of the look-and-feel elements of an SESM web application. The chapter discusses these topics:

The SESM software uses Java resource bundles and properties files for localization of text, labels on web-page controls, and messages. For information on localization and resource bundles, see Chapter 5, "SESM Internationalization and Localization."

Customizing an SESM Web-Application: An Overview

When you develop a Cisco SESM web application, three levels of web-application customization are possible: basic, advanced, and system integrator.

Basic SESM Customization

In some Cisco SESM deployments, you can use one of the sample SESM web applications as a starting point and accomplish look-and-feel modifications to the JSP pages without changing the preprogrammed SESM software. In this type of customization, the developer might change text, images, and the positioning of elements on a page. Basic customization might be appropriate when using SESM for Small-to-Medium-Size Enterprise (SME) wireless LAN hotspots that require simple look-and-feel changes to the web application or require some subset of SESM functionality, such as authentication.

This level of customization is also used by deployments whose business requirements correspond closely to the existing structure of a sample SESM web application, such as NWSP. This chapter discusses basic customization.

Advanced SESM Customization

Other SESM deployments will require use of the more advanced customization techniques. In this type of SESM web-application customization, you might add or remove JSP pages, or move elements from one JSP page to another. You might add a JSP-page dimension decorator for the SESM user-shape mechanism. You can accomplish some of the more advanced customizations by modifying the deployment descriptor file (web.xml) for the SESM web application. You accomplish other advanced customizations by creating new JSP pages. No Java servlet programming is needed. For information on advanced customization techniques, see Chapter 3, "Advanced SESM Customization."

System-Integrator Customization

Some service-provider deployments of SESM may need the professional services of system integrators to modify the functionality of the internal SESM software, such as the Java servlets for the SESM controls. This type of customization is beyond the scope of this guide and requires the involvement of Cisco technical assistance.

Changing the Look-and-Feel Elements

Basic SESM web-application customizations involve changing the look-and-feel elements to meet the service provider's brand requirements. The customizable JSP pages and look-and-feel elements allow developers to create web pages that incorporate artistic flexibility and meet corporate identity standards without requiring extensive JSP or Java programming expertise.

To meet the service provider's corporate standards, you can modify static markup language elements and images (template text) that appear in a template or JSP page. For example, you can change the static HTML elements for text and formatting. In these elements, the developer can:

- Change the background colors and background images used in the JSP pages
- Adjust the layout of the JSP pages
- Modify or replace icons, images, and graphical elements used in the JSP pages
- Customize a style sheet

If you are performing only basic look-and-feel customizations, you must avoid changing programmatic elements within the JSP pages. Do not change directives and code in scripting elements such as JSP expressions, scriptlets, and declarations.

Customizing some of the SESM functionality that is preprogrammed into the JSP pages can be accomplished by configuring the application. Other functionality changes may require modifying the code in the JSP pages. For information on these types of customization, see Chapter 3, "Advanced SESM Customization."

Using a Sample SESM Web Application

Each sample Cisco SESM web application, such as New World Service Provider (NWSP), includes a fully functional set of web components. Each sample web application uses the same SESM infrastructure. The simplest, fastest-to-implement approach to developing an SESM web application is to use the components in a sample SESM web application and then change the look-and-feel elements. This section provides an overview of the set of components in the sample SESM web applications and focuses on the NWSP web application.

The sample NWSP, PDA, and WAP web applications share many of the same components.

The sets of infrastructure components provided in the PDA and WAP sample web applications are identical to the NWSP set of infrastructure components. However, because the PDA and WAP web applications are designed for specific client devices, they use JSP pages that are different from the JSP pages of NWSP.

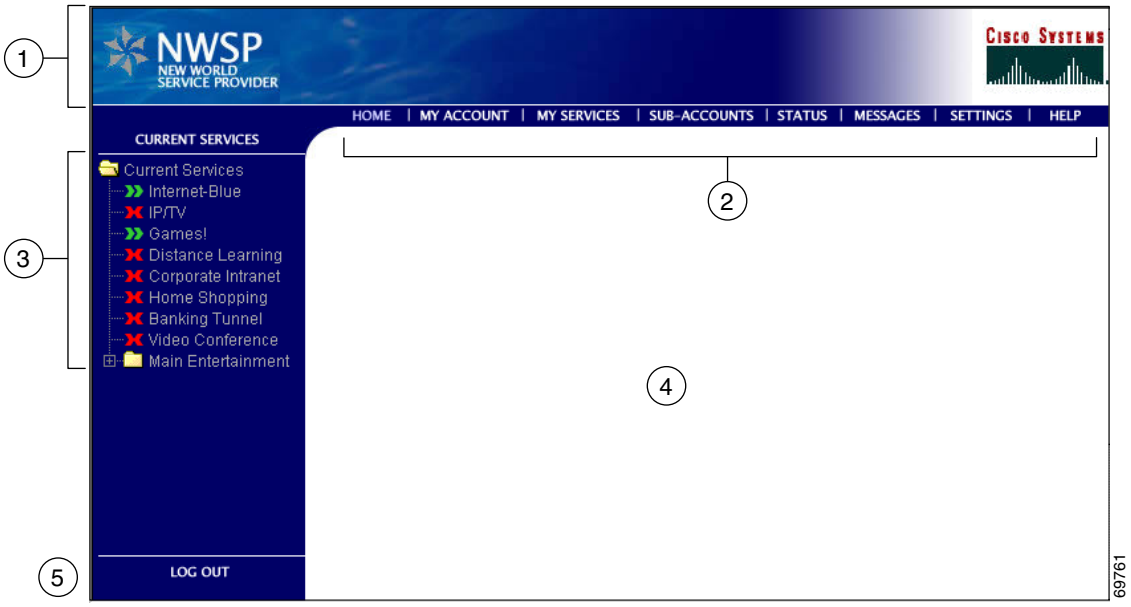
The sets of resources (Java resource bundles) used in the PDA and WAP sample web applications are very similar to the NWSP set of resources. The deployment descriptor files used for the NWSP, PDA, and WAP applications are also very similar.

For more detailed information on the elements that appear on the JSP pages associated with a sample SESM web application, see Chapter 4, “Sample SESM Web Applications.”

NWSP User Interface

The NWSP user interface provides a web portal for network services. The subscriber uses the web portal for subscribing to and selecting services, changing account details, creating subaccounts, and viewing session status and messages. Figure 2-1 shows the home page from the NWSP web application’s user interface.

Figure 2-1 NWSP Home Page



1	Banner
2	Navigation bar
3	Service list
4	Body
5	Log Out button

The Home page and many other pages in the NWSP web application are organized into five parts:

- **Banner**—One or more images for product and brand identification.
- **Navigation bar**—A set of buttons that link to other pages where the subscriber can perform tasks such as service subscription or view information such as service status.
- **Service list**—A tree of icons and associated service names that the subscriber uses to select subscribed services from a tree of services and service groups.
- **Body**—An area for the content provided by the NWSP web application. Forms for subscriber tasks such as account management, service subscription, and subaccount creation appear in the body area. (With NWSP, network services are displayed in a new browser window.)
- **Log Out button**—A button to log out of the NWSP web application.

JavaServer Pages

The sample SESM web applications are implemented in a set of JSP pages and Java servlets. To change the look and feel of an SESM sample web application, you need to become familiar with the elements that are present in the JSP pages provided in a sample SESM web application such as NWSP.

The JSP pages in the NWSP web application include a complete set of customizable images, background colors, icons, buttons, a navigation bar, and style sheets. For NWSP, the Fireworks graphics design tools were used to create the icons for the service list and the buttons for the navigation bar.

Banner Images and Background Colors

In the NWSP banner (see Figure 2-1), the images and background colors are used for branding. You can replace these images with company, product, and brand logos that are appropriate for your deployment.

Icons and Buttons

For the NWSP service list and navigation bar (see Figure 2-1), the icons and buttons are provided in various formats, including GIF, JPEG, and PNG.

- The GIF and JPEG image files are incorporated into the web pages. Their small file size makes the optimized GIF and JPEG files suitable for downloading. You can also use these files to edit the images. You can use Fireworks or another graphics design tool, such as Photoshop, to customize the images and embedded text.
- The PNG files are used when the web designer edits the images. Because PNG format is the native Fireworks file format, you can use Fireworks to customize the images and embedded text. PNG files can also be read by other graphics applications, such as Photoshop.

Buttons, such as the Log Out button and Cancel button, that do not appear in the NWSP navigation bar are defined once in a single JSP page or in a Dreamweaver template. You can customize each button in the one location where it is defined and have the change take effect in all JSP pages that use the button.

Navigation Bar

A Dreamweaver navigation bar (sometimes called a nav bar) is a set of buttons that appears on a series of related web pages and that provides a consistent mechanism for navigation between pages. For example, the NWSP web application contains a navigation bar (see Figure 2-1) below the banner on most of its pages.

When modifying a Dreamweaver navigation bar, you can use the existing NWSP buttons or create a new set of buttons. You can use Fireworks or another graphics design tool to change the text on the existing NWSP buttons. For more information on the navigation bar, see the "NWSP Navigation Bar" section on page 4-15.

Style Sheets

The NWSP web application uses Cascading Style Sheets for its presentation elements. The style sheets define classes for the textual elements that are used on the JSP pages. The style sheets allow designers to change fonts, margins, colors, and other aspects of style that control the layout and design of the NWSP web pages. The style sheets used by NWSP are located in the *install_dir*\nwsp\docroot\styles directory. In other SESM web applications such as PDA, there may be multiple brand-specific style sheets in different directories from which the SESM software dynamically selects one style sheet based on the subscriber's brand.

Dreamweaver Templates

The sample NWSP web application includes two Dreamweaver templates. These files have the suffix `.dwt` and reside in the /nwsp/docroot/templates directory. Dreamweaver templates can be very useful for customizing or maintaining a web application's JSP pages when many pages have the same layout. By modifying a template and then updating the JSP pages that use the template, you can change the look and feel of an entire set of pages very quickly.

When a JSP page is derived from a template, the JSP page has locked regions that cannot be edited and editable regions that can be edited. The intention with locked regions is that if changes are required on a locked region, the changes are made in the template file. After the changes are made to the template, all files that use the template can then be automatically updated to incorporate the changes. You modify the template and update all occurrences on the web site using the Dreamweaver **update** commands in the Modify > Templates menu.

The use of Dreamweaver templates and automatic updating is a recommended approach but not a requirement.

If changes are required to individual JSP pages (as opposed to in a Dreamweaver template), the part of the individual JSP page that you can modify appears between BeginEditable and EndEditable comments. For example, the main editable area in the mainTemplate.dwt is empty, allowing each JSP page to provide content that is appropriate for its purpose:

```
!-- #BeginEditable "main" -->
 
<!-- #EndEditable -->
```

For more information on templates, see the "NWSP Templates" section on page 4-10 and the Dreamweaver UltraDev documentation.

Developing an SESM Web Application

Depending on the service provider's business requirements for an SESM web application, the steps required to develop the application may vary. These steps are described in the following sections:

Tip One error that developers frequently commit when developing an SESM web application is that they do not perform the steps required *to compile* the JSP pages after they have modified them. The SESM sample web applications are installed with *precompiled JSP pages*. After changing the JSP pages, you must take a few simple steps to recompile them successfully. For information on compiling JSP pages, see the "JSP Compilation" section on page 2-8.

Defining the Business Requirements

The process of defining the business requirements for a specific service provider's SESM web application can be organized around the following questions:

- What functionality is required?
- What look-and-feel elements (icons, images, background colors, and style sheets) must be customized?
- Will SESM web pages be rendered to match one or more subscriber characteristics, such as device or brand?

- What types of localization (if any) are required?
  - If only English-language subscribers will use an SESM web application, the base set of components in a sample SESM web application may not require localization.
  - If subscribers use a single language other than English, a single web site localized for this language may be sufficient.
  - If subscribers use many languages, rendering SESM web pages localized for each subscriber's language and character set may be required.
- Do any application-specific messages need to be internationalized and localized?

Designing and Implementing an SESM Web Application

The design and implementation phases may involve one or more of the following tasks:

- Modifying the functionality of a sample SESM web application
- Customizing the look and feel of web elements such as icons, images, background colors, and style sheets
- Localizing web elements
- Creating additional locale-specific properties files
- Designing, implementing, and populating a sparse-tree directory
- Coding revised or new JSP-pages dimension decorators for the user-shape mechanism
- Configuring the web application deployment descriptor file
- Internationalizing and localizing exceptions

SESM Class Libraries

The Cisco SESM software provides several specialized Java class libraries that encapsulate the functionality required in an SESM web application. The SESM class libraries are distributed in the JAR (Java archive) files listed in Table 2-1. In the table, *install_dir* is the directory where the SESM software is installed, and *webapp* is a directory where a sample SESM web application, such as NWSP, is installed.

Table 2-1 JAR Files for an SESM Web Application

JAR File	Description
install_dir/lib/lib/ com.cisco.sesm.lib.jar	Classes for Java Management Extensions (JMX), JUnit utilities, and logging facilities used by an SESM web application.
install_dir/*webapp*/docroot/Web-inf/lib/ com.cisco.sesm.contextlib.jar	Classes for the SESM decorators and controllers, internationalization and localization, and tag libraries.
install_dir/*webapp*/docroot/Web-inf/lib/ jsp.jar	Classes for the precompiled JSP pages.
install_dir/*webapp*/docroot/Web-inf/lib/ sesm.jar	Classes for core and model services such as authentication, authorization, service connection, and billing.

The CLASSPATH environment variable must be set to *install_dir*/*webapp*/docroot/Web-inf/lib to tell the Java compiler the location of the com.cisco.sesm.lib.jar file. In the sample SESM web applications, the environment variable is set through the start script (for example, startNWSP.sh on UNIX and startNWSP.cmd on Windows), which is executed when the web application is started. Any JAR file, including com.cisco.sesm.contextlib.jar, jsp.jar, and sesm.jar files, can be found by the Java compiler if it resides in the web application's /Web-inf/lib directory.

Javadoc Documentation

The Cisco SESM software includes a full set of online documentation in Javadoc format. The Javadoc documentation includes information on all Java classes that implement the SESM software. If you are customizing the JSP pages of an SESM web application, the Java class descriptions for control servlets, decorator servlets, JavaBeans, and tag libraries are of particular interest. You can access the Javadoc by opening the file *install_dir*\docs\apidoc\index.html in a browser.

JSP Compilation

The use of precompiled JSP pages allows an SESM server that has the required Java Runtime Environment (JRE) to run the NWSP web application. If an SESM server has the JRE, it can run an SESM web application without having a Java 2 SDK installed.

The JSP pages in each sample SESM web application are precompiled and the resulting servlet classes are installed in the *install_dir**webapp*\docroot\Web-inf\lib\jsp.jar file. In the path name, *install_dir* is the directory where the SESM software is installed, and *webapp* is a directory where a sample SESM web application, such as NWSP, is installed.

A Java 2 SDK is not required to run an SESM web application. However, to perform development on the JSP pages, you *must* install the Java 2 SDK on the development machine. The Java 2 SDK is necessary for recompiling changed JSP pages. For information on the required Java 2 SDK, see the "Hardware and Software Requirements for Development" section on page 1-6.

Tip On machines that are used for SESM web application development, we recommend that you install the Java 2 SDK before you install the SESM software. In that way, the SESM installation program uses the Java 2 SDK in the SESM web application startup scripts, rather than a JRE.

To check whether a Java 2 SDK is installed on the development machine, look for a tools.jar file in the *JDK_install_dir*/lib directory, where *JDK_install_dir* is the location where you think the Java 2 SDK is located. If the tools.jar file is present, a Java 2 SDK is installed.

If you install the Java 2 SDK *before* installing SESM, read the "Recompiling and the web.xml File" section on page 2-9.

If you need to install the Java 2 SDK *after* installing SESM, read the "Installing a Java 2 SDK After Installing SESM" section on page 2-8 and the "Recompiling and the web.xml File" section on page 2-9.

Installing a Java 2 SDK After Installing SESM

If you install the Java 2 SDK *after* installing SESM, you must do the following:

- Ensure that the JDK_HOME environment variable points to the directory where you installed the Java 2 SDK.
- Edit the SESM application startup script to use the Java 2 SDK.

The SESM application startup scripts are named start*WEBAPP*.cmd on Windows-based installations and start*WEBAPP*.sh on UNIX-based installations. In the script name, *WEBAPP* is the name of the web application. For example, the startup script for NWSP is startNWSP.cmd or startNWSP.sh.

In the startup script on a Windows-based installation, change the line that specifies the Java 2 SDK location (JDK_HOME) to point to the correct directory, and uncomment the two lines that check for a Java compiler (JAVAC).

```
rem find some java or other
set JDK_HOME=D:\jdk1.3

set JAVA=%JDK_HOME%\bin\java.exe
set JAVAC=%JDK_HOME%\bin\javac.exe

rem Note that we only need this if we are developing, so for now
rem we can comment the check out. Developers should remove these comments
rem if exist "%JAVAC%" goto gotjdkhome
rem echo JDK_HOME does not point to a valid JDK. JSPs will not be compiled.
```

In startup script on a UNIX-based installation, change the line that specifies the Java 2 SDK location (JDK_HOME) to point to the correct directory, and uncomment the four lines that check for a Java compiler (JAVAC).

```
# Check we can find a suitable version of the JDK
JDK_HOME=/usr/java1.2

JAVAC=$JDK_HOME/bin/$JAVACEXE
JAVA=$JDK_HOME/bin/$JAVAEXE

# Note that we only need this if we are developing, so for now
# we can comment the check out. Developers should remove these comments
#if [ ! -x $JAVAC ]
#then
#        echo WARNING: JDK_HOME does not point to a valid JDK. JSPs can not be compiled.
#fi
```

Recompiling and the web.xml File

The web.xml file in *install_dir**webapp*\docroot\Web-inf is the SESM web application deployment descriptor file. This file defines how the Jetty web server finds the JSP pages. The default web.xml file for each sample SESM web application specifies that the web server uses the precompiled JSP pages. To compile modified JSP pages, use the web.recompile.xml file in place of the default web.xml file.

Note Until you perform the following steps, changing the JSP pages in *install_dir*/*webapp*/docroot has no effect on the HTML pages that the NWSP web application displays.

To recompile modified JSP pages in the NWSP web application, you must do the following:

Step 1 Stop the web server.

Step 2 In the *install_dir*\nwsp\docroot\Web-inf directory, rename the web.xml to web.xml.bak.

Step 3 In the *install_dir*\nwsp\docroot\Web-inf directory, rename web.recompile.xml to web.xml.

Step 4 Restart the web server.

Demo Mode

You can install or configure the SESM software for demonstration mode (Demo mode) to observe how the NWSP web application works. In Demo mode, the NWSP web application can demonstrate most SESM functionality. For information on installing or configuring for Demo mode, see the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide*.

Note The sample PDA and WAP demo applications also work in Demo mode. However, because the PDA and WAP applications demonstrate only the subset of SESM functionality that is appropriate for their targeted devices, this description of Demo mode focuses on the NWSP web application.

Demo mode is also useful during some phases of web-application development because this mode does not require other system components, such as a configured SSG.

- If the web designer modifies the look and feel of the NWSP web application, the changes can be viewed in Demo mode to observe the results.
- If the web developer changes programmatic pieces of a JSP page, many changes in web-application behavior can be initially tested in Demo mode to verify the results.

For the NWSP sample application, Demo mode uses a Merit RADIUS file for subscriber, service, and service group information. By default, the Merit RADIUS file is named demo.txt and resides in the *install_dir*\nwsp\config directory. A Merit RADIUS file is an ASCII file that you can modify using a text editor. Using Demo mode, you can observe how changes to a subscriber, service, or service group profiles affect the NWSP web application.

Tip If you make changes to the demo.txt file, you will not be able to observe the changes until the web server is stopped and restarted.

Captive Portal and Demo Mode

The full set of behaviors of the Captive Portal solution cannot be observed in Demo mode because a configured SSG and its TCP Redirect feature are required to make the solution work. For this reason, the Captive Portal software is not installed in a Demo mode installation of SESM.

To run Captive Portal in Demo mode, you must install the Captive Portal software in RADIUS or LDAP mode, and then start the Captive Portal web application using the **-mode** option. For example:

```
startCAPTIVEPORTAL.sh -mode Demo
```

With Captive Portal running in Demo mode, you can use simulated HTTP requests to view the behavior of the Captive Portal servlet and the content web applications. For example, you could send a simulated HTTP request to the message portal servlet (`MessagePortalServlet`) if you included the required query-string parameters (such as `CPURL` and `CPDURATION`) in the request. For information on the Captive Portal solution and the query-string parameters, see the "Captive Portal Web Application" section on page 4-23.

RADIUS Mode Functionality

To simulate RADIUS mode functionality (such as service selection), the subscriber, service, and service-group profiles in demo.txt use the standard RADIUS attributes and vendor-specific RADIUS attributes (VSAs). The attributes are described in the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide*. For example, the following profile from demo.txt is for the subscriber radiususer:

```
radiususer Password = "cisco"
        Service-Type = Framed-User,
        Account-Info = "Ainternet-blue",
        Account-Info = "Ninternet-blue",
        Account-Info = "Niptv",
        Account-Info = "Ngames",
        Account-Info = "Ndistlearn",
        Account-Info = "Ncorporate",
        Account-Info = "Nshop",
        Account-Info = "Nbanking",
        Account-Info = "Nvidconf"
        Account-Info = "Hhttp://www.spiderbait.com"
```

DESS Mode Functionality

To simulate DESS mode functionality (such as subscriber self-subscription, account management, and subaccount creation), the allowed subscriber profile attributes have been extended so that the NWSP web application can demonstrate DESS mode features:

- Different types of subscriber privileges (for service selection, service subscription, account management, and subaccount creation)
- Account attributes with X.500 information (title, given name, surname, and so on)
- Services and service groups that are available for subscription but not yet subscribed
- Parent account names for subaccounts

The extended set of attributes like other RADIUS attributes used by an SESM web application are read-only. However, NWSP in Demo mode does correctly simulate DESS mode functionality. For example, in Demo mode, if a subscriber adds or deletes a subscription, the list of subscribed services on the My Services page and in the service list are dynamically updated. The changes persist until the web server is stopped. The NWSP web application does not modify attributes in the Merit RADIUS file.

Note The extended set of subscriber profile attributes are allowed for Demo mode only. The extended set of attributes are not valid vendor-specific attributes (VSAs). They are not valid in RADIUS mode or LDAP mode and are not recognized by the SSG. They should not be added to the RADIUS dictionary.

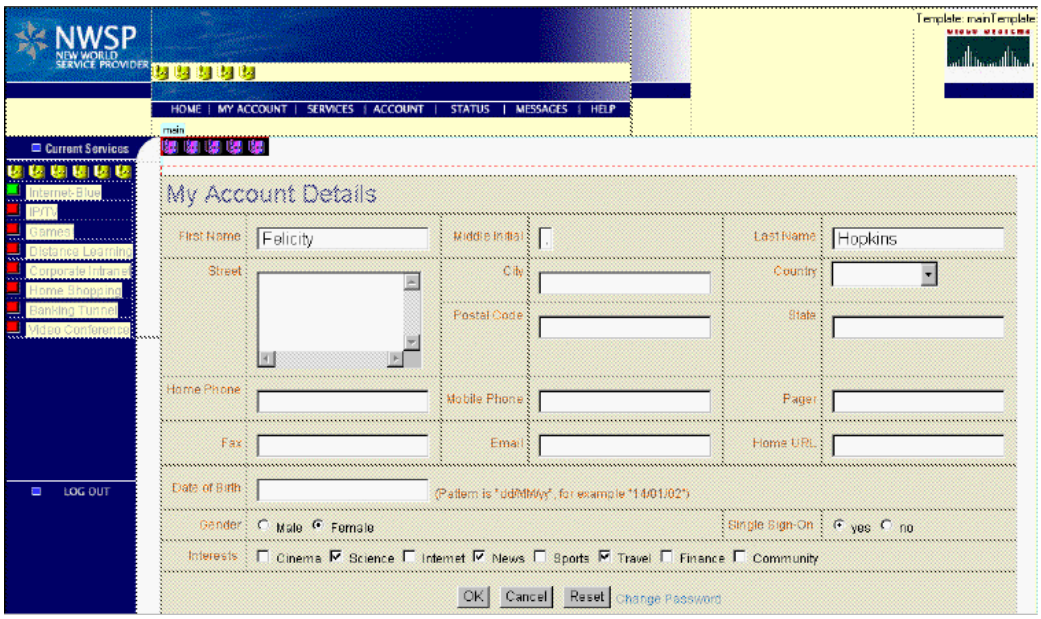
In NWSP, the demo.txt file contains two subscriber profiles for simulating DESS mode: ldapuser1 and ldapuser2. These subscriber profiles provide examples of the attribute extensions. For detailed information on the extended set of attributes for demonstrating DESS features, see the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide*.

Dreamweaver UltraDev 4 Live Data Window

When you are developing a Cisco SESM web application, you can use the Live Data window feature of Dreamweaver UltraDev 4 to make changes to your JSP pages in a live data environment. Unlike the Document window, which uses placeholders for dynamic content on a JSP page, the Live Data window shows the actual dynamic content that is generated by the JSP page.

Figure 2-2 shows how myAccount.jsp is displayed in the Live Data window. To display dynamic data, UltraDev runs the JSP page on the web server before displaying it in the Live Data window. In Figure 2-2, notice that the Live Data window displays services in the service list and any subscriber data (for example, first name and last name) in the subscriber profile.

Figure 2-2 Live Data Window



Note Editing in a live data environment is available starting with Dreamweaver UltraDev 4.

To configure and use the Live Data window for the NWSP web application, perform the following steps. The procedure assumes that the web server is running on the local machine and listening for requests for the NWSP web application on port 8080.

Step 1 Start Dreamweaver UltraDev 4.

Step 2 In the **Site** menu, click **New Site**.

Step 3 In the **Category** list, click **Local Info** and specify the following:

Site Name: NWSP

Local Root Folder: *C:\SESM_location*\nwsp\docroot\ (In this procedure, *C:\SESM_location* is the location where the SESM software is installed.)

HTTP Address: http://localhost:8080

Step 4 In the **Category** list, click **Remote Info** and specify the following:

Access: Local/Network

Remote Folder: *C:\SESM_location*\nwsp\docroot\

Step 5 In the **Category** list, click **Application Server** and specify the following:

Server Model: JSP 1.0

Scripting Language: Java

Page Extension: .jsp

Access: Local/Network

Remote Folder: *C:\SESM_location*\nwsp\docroot\ (for example)

URL Prefix: http://localhost:8080/user=golduser/device=pc/locale=en/shape/

In URL Prefix, the values given after the port number are test decorator values that use URL-to-servlet mapping to specify the user, device, and locale. For information on using test values, see the "Using Test Decorators" section on page 2-14.

Note To use the test decorators specified in the URL Prefix, copy the contents of the web.dev.xml file to the end of the web.xml file used by the SESM web application, and then save the modified web.xml file. Restart the web server. The web.dev.xml file is located in the *install_dir*\nwsp\config directory.

Step 6 Click **OK**.

Step 7 To use the Live Data window to view a JSP page:

a. Double-click the name of the JSP page.

b. In the Document window that displays the JSP page, from the **View** menu, click **Live Data**.

UltraDev displays the actual dynamic content that is generated by the JSP page. In the Live Data window, you can edit the page's layout and JSP-page content. For information on editing in a live data environment, see *Using Dreamweaver UltraDev*, which is available on the web at:

http://www.macromedia.com/support/ultradev/documentation/using_ultradev_4.html

General Web Development Considerations

Some general web development considerations for a Cisco SESM web application include:

- We recommend the technique in which the web application redirects a `POST` request to a `GET` request. The Cisco SESM software has no mechanisms that support or prevent this technique.
- The decoration JSP pages and include files must not write and flush any characters to the `ServletOutputStream`. If either does, the forwarding HTTP request throws an `IllegalStateException`. This restriction is imposed by the servlet container.
- As is usual with any web application, all references to a web resource from a web page must be relative.

A web application can be deployed with a prefix, which is specified in the web.xml file. An absolute reference to another web resource within the same web application must include the web application prefix. However, the web application has no knowledge of the web application prefix. Therefore, references to web resources must be relative.

Debugging an SESM Web Application

If an SESM web application is not operating as expected, you can use the SESM logging utility to change the details of the information reported in the log files to diagnose a problem. For detailed information on logging and debugging mechanisms available for an SESM web application, see the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide*.

The following hints may help with debugging an SESM web application:

- Most web servers can be set up so that the servlet code generated from JSP pages is saved and available for examination. With the Jetty web server, when the JSP pages are compiled, the servlet code is put in a temporary directory that is determined by the machine's Java Virtual Machine (JVM). Normally, the temporary directory is /tmp or /usr/tmp on UNIX, and C:\TEMP on Windows NT or Windows 2000. The servlet code can be examined in the temporary directory until it is deleted by the operating system's normal cleanup mechanism. For information on where servlet code is located with other web servers, refer to the documentation from your web server vendor.
- With the Internet Explorer browser, when an error occurs in the dynamic portion of a JSP page, Internet Explorer displays an alternative "friendly" (and less helpful) HTTP error message if the Show friendly HTTP error messages box is checked. To view server-generated error messages, make sure this box is not checked in the Advanced Tab for Internet Options, which you access through the Tools menu.
- Some SESM web application files such as web.xml and the properties files for resource bundles are read once when the web application is started. Changes to these files have no effect until the web application is stopped and restarted.

Using Test Decorators

The SESM software includes a number of Java servlets called *test decorators* that are very useful for development and testing of an SESM web application. The test decorators allow the SESM developer to set specific values for

- User name and password
- Locale language, country, and variant
- Other dimensions of the user shape such as brand and device

The web.dev.xml file has declarations for a number of test decorators preconfigured with some common values. For NWSP, the web.dev.xml file is located in the *install_dir*\nwsp\config directory. You can add to or modify the test values in web.dev.xml to meet your development and testing needs. For information on user-shape decoration and decorators, see the "SESM Software Concepts" section on page 3-6.

Note To use the test decorators, copy the contents of the web.dev.xml file to the end of the web.xml file used by the SESM web application, and then save the modified web.xml file. Restart the web server.

Each test decorator has an associated URL-to-servlet mapping that allows you to specify the test values in the URL when requesting an SESM web page. When the URL for a web application resource is preceded by URL-mapped test decorators, the corresponding test decorator servlets are invoked. Consider the following URL requesting myAccount—a control servlet for the My Account page:

http://localhost:8080/**user=golduser/device=pda/locale=de/**myAccount

When the preceding URL is used in demonstration mode, test decorator servlets are invoked and do the following:

- Authenticate the user name and password for golduser
- Set the `device` dimension of the user shape to `pda`
- Set the `locale` dimension of the user shape to `de` (Germany)

The following sections provide information on the test users, locales, and `device` and `brand` dimensions that are preconfigured in the web.xml file. For information on how you can modify and add to the set of test values and for information on the test decorator servlets, see Appendix B, "SESM Utility Servlets Quick Reference".

Test Users

Table 2-2 lists some of the test users, passwords, and URL mappings that are preconfigured in the web.dev.xml file. For each URL mapping, the `TestUserDecorator` servlet sets the user name and password to the values shown in the table. `TestUserDecorator` then automatically attempts to authenticate with the user name and password. In Demo mode, the SESM software authenticates against the subscriber profiles defined in the Merit RADIUS file demo.txt. The demo.txt file resides in the /config directory of the SESM web application.

Table 2-2 Test Users: Names and Password

User Name and Password	URL Mapping
User name: user1 Password: cisco	/user=user1/*
User name: user2 Password: cisco	/user=user2/*
User name: user31 Password: cisco	/user=user31/*
User name: user42 Password: cisco	/user=user42/*
User name: user43 Password: cisco	/user=user43/*
User name: user44 Password: cisco	/user=user44/*
User name: golduser Password: cisco	/user=golduser/*
User name: subgolduser Password: cisco	/user=subgolduser/*

Test Locales

Table 2-3 lists some of the test locales, associated values, and URL mappings that are preconfigured in the web.dev.xml file. For each URL mapping, the `LocaleDecorator` servlet sets the `locale` dimension to the value shown in the table.

Table 2-3 Test Locales

Locale	Value	URL Mapping
Australia	`au`	/locale=au/*
France	`fr`	/locale=fr/*
Germany	`de`	/locale=de/*

Table 2-3 Test Locales (continued)

Locale	Value	URL Mapping

Great Britain	`en`
	`gb`
	`uk`
		/locale=en/*
		/locale=gb/*
		/locale=uk/*

Italy	`it`	/locale=it/*
Portugal	`pt`	/locale=pt/*
Spain	`es`	/locale=es/*
Sweden	`se`	/locale=se/*
USA	`us`	/locale=us/*

Test Devices and Brands

Table 2-4 lists some of the test devices, brands, associated values, and URL mappings that are preconfigured in the web.dev.xml file. For each URL mapping, the `TestDimensionDecorator` servlet sets the `device` or `brand` dimension to the value shown in the table.

Table 2-4 Test Devices and Brands

Device or Brand	Value	URL Mapping
WAP device	`wap`	/device=wap/*
PDA device	`pda`	/device=pda/*
PC device	`pc`	/device=pc/*
XML device	`xml`	/device=xml/*
NWSP brand	`nwsp`	/brand=nwsp/*
Unstyled brand	`unstyled`	/brand=unstyled/*
Gold brand	`gold`	/brand=gold/*
Silver brand	`silver`	/brand=silver/*
Bronze brand	`bronze`	/brand=bronze/*
No brand	an empty string	/brand=/*

Using Device Simulators for WAP and WML

WAP phone simulators are very useful for development and testing an SESM web application that generates WAP content using Wireless Markup Language (WML). The sample WAP web application is designed for providing content to a subscriber who is using a WAP phone. WAP phone simulators allow you to view WAP content by using the HTTP protocol to access the JSP pages of an SESM web application. Figure 2-3 shows the home page and service list of the WAP web application as it appears on a Nokia simulator.

Figure 2-3 WAP Phone Simulator

When the HTTP protocol mode is used to access an SESM web application, you request pages from an SESM web application in the usual manner. For example, to log on to SESM, you specify the following URL: http://*web_server*:8080, where *web_server* is the IP address of the Jetty web server.

The NWSP and WAP web applications recognize many WAP phone simulators, including the Nokia Mobile Internet Toolkit simulator, the Openwave UP.Simulator (version4.0 and 4.1), and the WinWAP Pro simulator. The NWSP and WAP web applications set the `device` dimension of the user shape to `wap` when these simulators are the client device.

WAP phone simulators that are available on the web include:

- Nokia Mobile Internet Toolkit contains two simulators and is available at Forum Nokia:

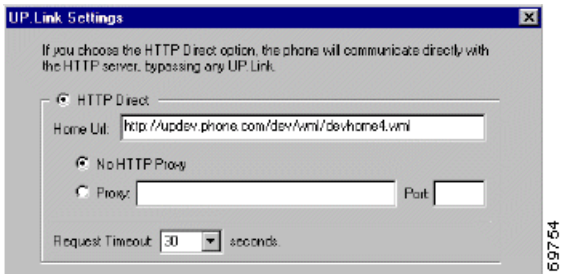
  http://www.forum.nokia.com/

- UP.SDK for WML from Openwave Systems Inc. contains a simulator and is available at:

  http://developer.openwave.com

To use the UP.Simulator and the HTTP protocol to access an SESM web application, the UP.Simulator must be configured to use HTTP Direct as the UP.Link Setting. To do this, start the UP.Simulator, click UP.Link Settings from the Settings menu, and then click HTTP Direct in the UP.Link Settings dialog box (Figure 2-4).

Figure 2-4 UP.Link Settings for the UP.Simulator



Managing an SESM Web Site

Dreamweaver has a number of features that may be useful for developing an SESM web application. For example, if a Dreamweaver template is modified, you can automatically update all files in a web site that use the template. To use some Dreamweaver features, you must define a web site in Dreamweaver.

To define an SESM web application as a site, do the following:

Step 1 From the **Dreamweaver Site** menu, choose **New Site** and specify the required information.

Step 2 In the **Local Info** dialog box, specify the */webapp/docroot* directory of the SESM web application as the Local Root Folder.

Step 3 In the **Site Map Layout** dialog box, specify home.jsp as the Home Page.

After the site is defined in this manner, Dreamweaver creates a useful Site Files view. However, because home.jsp is designed to contain few links to other JSP pages, the Site Map view is not very useful.

For detailed information on setting up and managing a site, refer to the Dreamweaver documentation.

The Dreamweaver Preview in Browser feature is of limited use with an SESM web application. For example, in the NWSP web application, most JSP pages require some form of input. Without the input, the web browser displays an error page. If you are using Dreamweaver UltraDev 4, the Live Data window feature is an excellent alternative to the Preview in Browser feature. The Live Data window allows you to view dynamic data in a sample SESM web application. For information on configuring the Live Data window, see the "Dreamweaver UltraDev 4 Live Data Window" section on page 2-11.

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

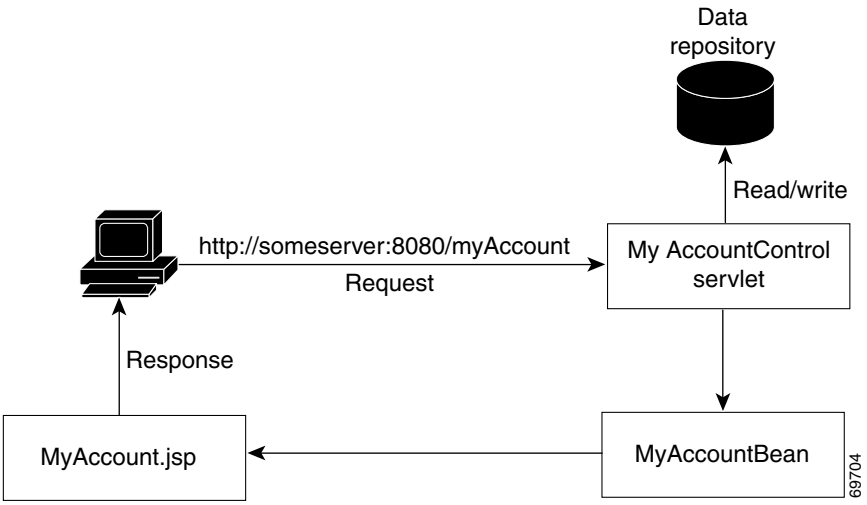
Each control can have many views. For example, there is one `MyAccountControl` but multiple JSP files can be used for the view myAccount.jsp. Each of the JSP pages for myAccount.jsp might provide content tailored for a different device (for example, WAP phone, PDA, or PC). In effect, the `MyAccountControl` 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

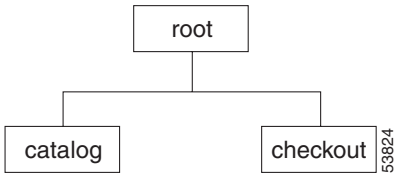
These two hierarchies are developed independently of each other. However, all web resource files are located within one web application because you combine the two hierarchies into one large hierarchy whose root is the web application. For this discussion, a *web resource* might be a JSP page, HTML file,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

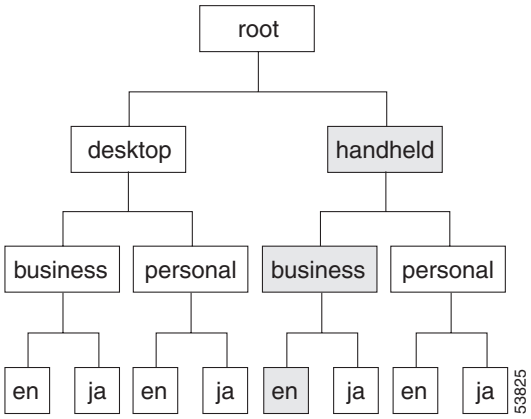
- Devices: desktop and handheld
- Brands: business and personal
- Locales (languages): English (en) and Japanese (ja)

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.

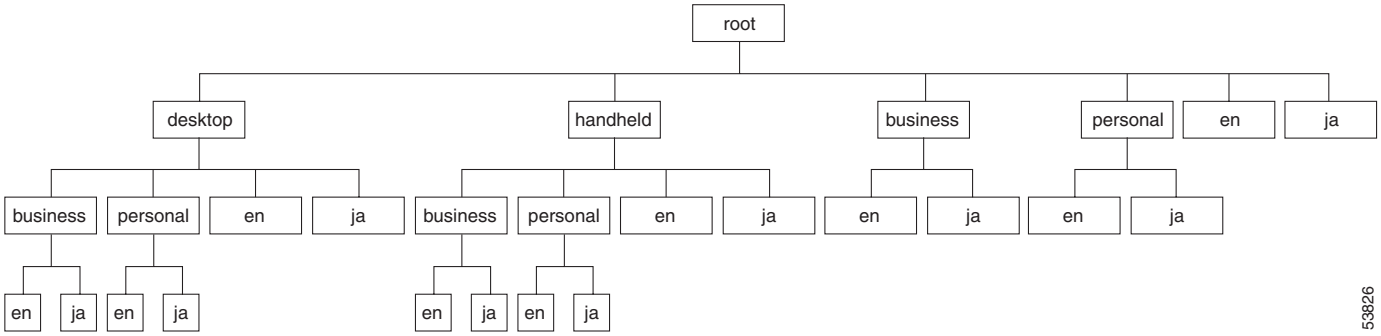
- In general, the more frequently a value appears in the user-shape directory hierarchy, the further down in the hierarchy it is located. As an example, in Figure 3-3 the values of the first dimension (desktop and handheld) appear only once in the hierarchy, but the values of the last dimension (en and 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.

CHAPTER 4

Sample SESM Web Applications

This chapter provides information on the sample SESM web applications and solutions and describes how a developer can use and modify the sample web components. The Cisco SESM software includes the following sample web applications and solutions:

The New World Service Provider (NWSP) web application is a fully featured example of the technology that SESM provides. This chapter provides detailed information on NWSP and its components.

The PDA and WAP web applications and the Captive Portal solution are designed for specific purposes and illustrate a subset of SESM web application technology. This chapter provides overview information on the PDA and WAP applications and Captive Portal.

General Considerations for Sample SESM Web Applications

The following general considerations apply to this chapter's descriptions of the sample SESM web applications and solutions.

- Before you read this chapter, read Chapter 2, "Basic SESM Customization and Development" and Chapter 3, "Advanced SESM Customization" for an explanation of SESM components and techniques that are, in general, used in all SESM sample web applications.
- Depending on the SESM software that is used, a deployed SESM web application can be configured for one of two modes:
  - *RADIUS mode*—Service and subscriber information is stored in a RADIUS server.
  - *LDAP mode*—Service, subscriber, and policy information is stored in an LDAP-compliant directory, which is accessed with the DESS application programming interfaces.
- The set of web components used for an SESM web application varies depending on the configuration (RADIUS mode or LDAP mode). Where it is required, the descriptions of components in this chapter indicate the mode in which each component is used.

New World Service Provider Web Application

The sample New World Service Provider (NWSP) web application contains all of the components required for a fully functional web portal for network services. The developer can use the NWSP web components as a starting point for designing and creating an SESM web application. This section provides information on the NWSP web application and describes how a developer can use and modify the NWSP web components.

NWSP User Interface

For information on the NWSP user interface, see the "NWSP User Interface" section on page 2-3.

NWSP Customization Based on Subscriber Characteristics

The look and feel of an SESM user interface can be customized for each subscriber. When the HTTP request is received, the SESM web application has an organization (the sparse-tree directory structure) and an infrastructure (the user-shape mechanisms) that allow the page returned in the response to be tailored for the individual subscriber. For example, if the subscriber's browser language is English and receiving device is a desktop PC, the response can be rendered in English using HTML. If another subscriber's browser language is German and the receiving device is a WAP phone, the response can be rendered in German using Wireless Markup Language (WML).

For information on customization and localization, see Chapter 3, "Advanced SESM Customization," and Chapter 5, "SESM Internationalization and Localization."

NWSP Functionality

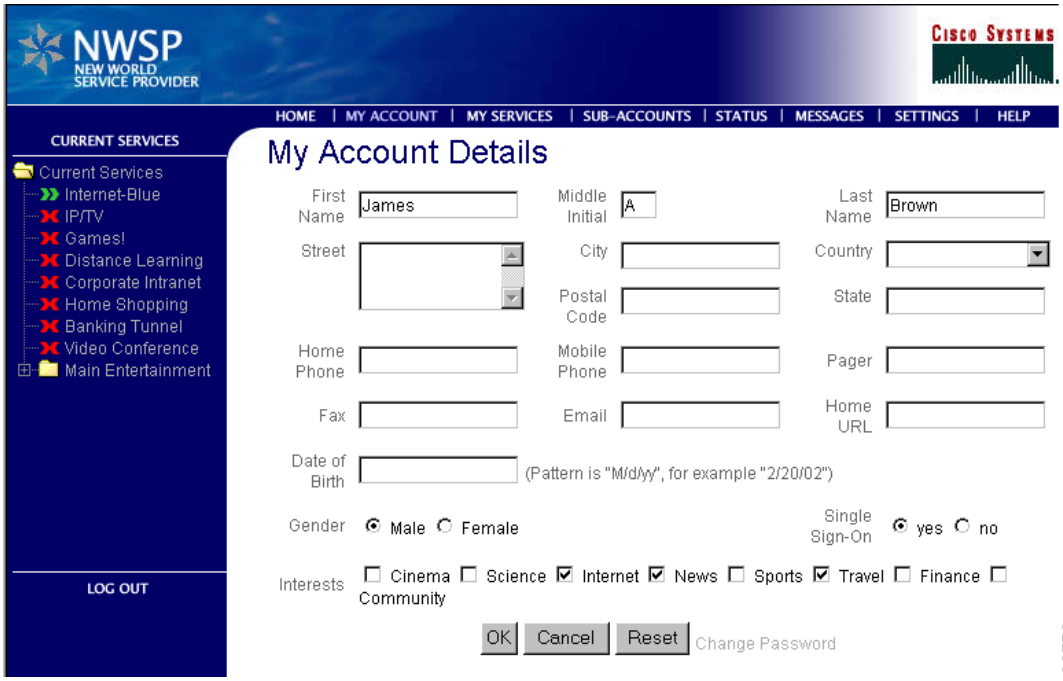
The sample NWSP web application provides a set of functionality that is typical of many directory-enabled SESM applications. The subscriber logs on to the web portal with a user name, password, and for 3-key authentication, a telephone number. The subscriber can then do the following:

- Subscribe to or unsubscribe from network services that are authorized
- Connect to or disconnect from services that are subscribed
- Change account details, such as address information and passwords
- Create subaccounts for other family members
- View the status of service connections
- View system messages

The features related to account management are possible only with an SESM web application that is operating in LDAP mode. The NWSP web application includes the required logic to determine the permissions that were granted to a subscriber and to generate the appropriate content. For example, if a subscriber has the required permissions to create subaccounts, the NWSP web application displays the Accounts button in the navigation bar, and the subscriber can create subaccounts.

Each button at the top of the user interface is linked to a JSP page that implements the functionality for the specific task or set of tasks. As an example, the My Account button is linked to myAccount.jsp, which generates the content for the account management page (Figure 4-1).

Figure 4-1 Account Management Page



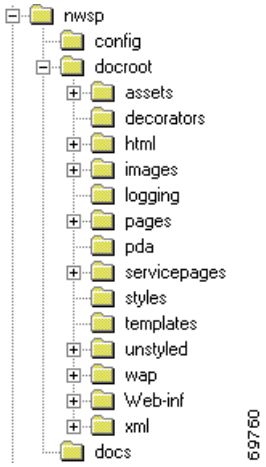
NWSP Directory Hierarchy

After the Cisco SESM software is installed, the NWSP web application is located in a structured hierarchy of directories. As with any web application, the root of this hierarchy is the document root for serving the NWSP web pages to the subscriber. In this directory hierarchy, the \Web-inf directory contains items related to the web application that are not in the document root. That is, the files and directories in \Web-inf are not part of the public document tree from which files can be directly served to the client.

When the Cisco SESM software is installed, the NWSP web application's hierarchy of directories is located below the *install_dir*\nwsp directory (see Figure 4-2).

Note The obsolescent NWSP web application components from SESM Release 3.1(1) are in the nwsp311 directory. Do not use any of the programmatic components from the nwsp 311 directory for a new SESM web application.

Figure 4-2 NWSP Directories



The NWSP directories contain the complete set of files required for the web application and include the PNG source files required to customize the Fireworks images and buttons. The NWSP directories and files are as follows.

Tip The following directories and their files are used for application development and are not required in a deployed SESM web application: \docroot\assets, \docroot\templates, and \docs.

config

Contains the web application configuration file nwsp.xml. For information on the configuration files, see the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide*.

docroot

Is the web application's document base.

docroot\assets

Contains the PNG source files from which the GIF and JPEG images were made. You can customize the images in the PNG files using Fireworks or another graphics tool.

docroot\decorator

Contains the JSP pages used for decorating the user's shape. For information on user-shape decoration, see the "Decorating a User Shape" section on page 3-16.

docroot\html

Contains files for specific versions of HTML: 3.0 and 4.0, such as a set of service-list files for HTML 4.

docroot\images

Contains the GIF and JPEG images for the NWSP user interface. The \docroot\images\serviceList directory contains the icons for service status that appear in the service list when the value of the brand dimension is not `nwsp`.

docroot\logging

Contains files that are used when logging entries are made from a JSP page.

docroot\nwsp\serviceList

Contains GIF images for the NWSP user interface. The \docroot\nwsp\images\serviceList directory contains the icons for service status that appear in the service list when the value of the brand dimension is `nwsp`.

docroot\pages

Contains the JSP pages that contain the web-application content. The \l10n directory contains JSP pages that provide information on the current localization context and the locales and timezones supported by the JRE. For information on the JSP pages used in NWSP, see the "NWSP JavaServer Pages and Servlets" section on page 4-6.

docroot\pda

Contains the files that are used when the subscriber's device is a PDA.

docroot\servicepages

Contains images for services that the NWSP web application uses in Demo mode.

docroot\styles

Contains the Cascading Style Sheets sesm.css and serviceList.css.

docroot\templates

Contains the Dreamweaver templates. For information on the NWSP templates, see the "NWSP Templates" section on page 4-10.

docroot\unstyled\pages

Contains files that are currently not used.

docroot\wap

Contains files that are used when the subscriber's device is a WAP phone.

docroot\Web-inf

Contains various Java-related NWSP components:

- Tag library descriptor (.tld) files for the NWSP tag libraries are in \Web-inf. For information on the .tld files and using a tag library, see the "Configuring a Tag Library" section on page A-1.
- The web application's deployment descriptor file, web.xml, is in \Web-inf. For information on this file, see the "Using the SESM Deployment Descriptor File" section on page 3-29.
- The \lib directory contains the Java archive (JAR) files for some SESM classes. For information on the JAR files required for an SESM web application, see the "SESM Class Libraries" section on page 2-7.
- The \classes directory contains the NWSP properties files. For information on localization and properties files, see Chapter 5, "SESM Internationalization and Localization."

docroot\xml\pages

Contains files that are used for an XML-based view for the service list control.

docs

Contains the Javadoc files for the NWSP classes. For information on the Javadoc files, see the "Javadoc Documentation" section on page 2-8.

NWSP JavaServer Pages and Servlets

The NWSP web application includes a set of JSP pages and servlets that generate content for the web pages and perform other tasks, such as authentication, SESM session handling, and service selection and subscription. The JSP pages contain the elements that the developer modifies for the specific requirements of the service provider. No servlet programming is required.

This section provides guidance on the JSP pages that generate the content for NWSP. For information on the architecture of an SESM web application and the servlets and decorator JSP pages used in NWSP, see Chapter 3, "Advanced SESM Customization."

After the subscriber logs on to the NWSP user interface, home.jsp is the home page for the web application. From the home page, the subscriber can control service subscription and selection and, in LDAP mode, can perform account-management, self-subscription, and subaccount functions. The navigation bar at the top of the home.jsp links the subscriber to these capabilities.

Modularized JSP Pages

In NWSP, many JSP pages that generate content are modularized into two pieces: a wrapper JSP page and a body JSP page.

Wrapper JSP Pages

Each *wrapper JSP page* contains standard functionality found on many JSP pages such as:

- JavaScript that must appear in the `<head>` section
- The NWSP banner
- The navigation bar
- The service list

For example, subscriptionManage.jsp is a wrapper JSP page containing JavaScript in its `<head>` section, the NWSP banner, the navigation bar, and the service list.

Most content for the wrapper JSP pages comes from the Dreamweaver template mainTemplate.dwt. If you are not using Dreamweaver as your HTML editor or your SESM web application does not use templates, having common elements like the JavaScript and banner isolated in a separate wrapper file should make it easier to modify these elements.

Body JSP Pages

Each *body JSP page* contains the functional elements (for example, an HTML form) that appear in the `<body>` section. The subscriber uses the functional elements to perform a task that is unique to the JSP page. Each body JSP page is included into the corresponding wrapper JSP page. The subscriptionManageBody.jsp page, which is included into the body of the wrapper page subscriptionManage.jsp, contains the functional elements that the subscriber uses to subscribe or unsubscribe from services.

User-Interface Control Pages

In NWSP, the user-interface controls for submitting, canceling, and resetting a form are also modularized in their own JSP pages. The appearance of these controls can be a button, image, or link depending on the value of the `type` attribute. For example, in submitButton.jsp, the `type` attribute specifies a button that users can click to submit the form's contents to the web server:

```
<input type="submit" value="<l10n:resource key='OK'/>">
```

As shown in the preceding example, the label (the `value` attribute) on the Submit button is localizable through the use of the `resource` tag and NWSP resource bundle property files. For information on the use of resource bundles, see the "Using Resource Bundles" section on page 5-3.

JSP Pages for Service Selection

Table 4-1 lists JSP pages for service selection and other NWSP content that is available in both RADIUS mode and LDAP mode. They are located in the \nwsp\docroot\pages directory.

Table 4-1 NWSP JSP Pages for Service Selection

JSP Page	Description
accountLogoff.jsp	After the subscriber clicks the Log Out button, asks the subscriber to confirm or cancel session termination.
accountLogon.jsp accountLogonBody.jsp	Allows a subscriber to log on to the NWSP web application. The subscriber's user name and password are authenticated by the SESM software.
accountLogon3Key.jsp accountLogon3KeyBody.jsp	Allows a subscriber to log on to the NWSP web application when 3-key authentication is used. When 3-key authentication is used, the subscriber credentials include a user name, password, and telephone number.
confirmBody.jsp	Displays a message and asks the subscriber to confirm or cancel a specific action by clicking the OK or Cancel button.
cancelButton.jsp	Displays a button, image, or link that navigates to the URL specified in the request parameter `"url"`. The cancelButton.jsp page is included into an HTML form on JSP pages where a cancel button appears.
help.jsp helpBody.jsp	Displays help information. The help information is for the demonstration mode NWSP.
home.jsp	Displays the NWSP home page. See Figure 2-1 on page 2-4.
message.jsp	Displays a message and asks the subscriber to click OK.
messages.jsp messagesBody.jsp	Displays messages associated with the current session.
navbar.jsp	Contains the NWSP navigation bar. For more information, see the "NWSP Navigation Bar" section on page 4-15.
serviceList.jsp serviceListGroup.jsp serviceListService.jsp	Displays the service list, a tree of subscribed services and service groups. For more information, see the "NWSP Service List" section on page 4-12.

Table 4-1 NWSP JSP Pages for Service Selection (continued)

JSP Page	Description
serviceLogon.jsp	Allows a subscriber to enter a user name and password when logging on to a service.
status.jsp statusBody.jsp	Displays the status of each connected service: • User name • Service description from the service profile • Service status • Connected as—the user name specified for a service that requires authentication • Elapsed connect time • Packets sent

Standard and Secure Mode

The accountLogonBody.jsp and accountLogon3KeyBody.jsp pages include Standard | Secure hyperlinks below the Log In button. These links let the user choose either standard mode or secure mode. On the logon page that accountLogon.jsp or accountLogon3KeyBody.jsp displays, if the subscriber is currently using secure mode, only the standard mode link is available. Similarly, if the subscriber is currently using standard mode, only the secure mode link is available.

When the subscriber logs in using secure mode, the SESM web application uses Secure Sockets Layer (SSL) encryption. The password that the subscriber enters is encrypted by the HTTP client before it is sent to the HTTP server where the SESM web application resides. The HTTP server decrypts the password. The encryption and decryption occurs for all content that passes between the client and the server, not just for the password.

If the service provider does not require SSL or does not have a certificate, the developer removes the Standard | Secure elements in the accountLogonBody.jsp and accountLogon3KeyBody.jsp pages. In addition, the deployer removes the Jetty web server's SSL listener, which is configured in the \nwsp\config\nwsp.xml file. For more information on SSL and security, see the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide*.

JSP Pages for Service Subscription, Account Management, and Subaccount Creation

Table 4-2 lists the JSP pages for service subscription, account management (subscriber self-care), and subaccount creation. These pages are used only with LDAP mode and are located in the \nwsp\docroot\pages directory.

Table 4-2 NWSP JSP Pages for Service Subscription, Account Management, and Subaccount Creation

JSP Page	Description
accountPassword.jsp accountPasswordBody.jsp	Displays a form for creating a new password.
accounts.jsp	Displays a form that allows the user to create and delete subaccounts.

Table 4-2 NWSP JSP Pages for Service Subscription, Account Management, and Subaccount Creation (continued)

JSP Page	Description
myAccount.jsp myAccountBody.jsp	Displays a form that allows a subscriber to change account information such as an address, telephone number, email address, and so on.
resetButton.jsp	Displays a button, image, or link that resets the enclosing form. The resetButton.jsp page is included into an HTML form on JSP pages where a Reset button appears.
subaccountConfirm.jsp subaccountConfirmBody.jsp	Displays and asks the subscriber to confirm changes to subaccounts specified in subaccountSubscriptions.jsp.
subaccountSubscriptions.jsp subaccountSubscriptionBody.jsp	Displays a form that allows a subscriber to modify subaccount details, such as subscribed and blocked services and, if applicable, the user name and password for a service.
subscriptionConfirm.jsp subscriptionConfirmBody.jsp	Displays and asks the subscriber to confirm changes to the subscriptions specified in subscriptionManage.jsp.
subscriptionManage.jsp subscriptionManageBody.jsp	Displays a form that allows a subscriber to subscribe to or unsubscribe from services, and to modify attributes associated with subscriptions. For example, a subscriber can specify whether the service is an auto-connect service, or can define the user name and password to access a service.
submitButton.jsp	Displays a button, image, or link that submits the enclosing form. The submitButton.jsp page is included into an HTML form on JSP pages where a Submit button appears.

In LDAP mode, the permissions defined for each subscriber determine whether the subscriber can subscribe to services, manage account details, and create subaccounts. The NWSP web application obtains the subscriber's permissions using the JavaBean `permissionBean` and then generates the NWSP user interface based on the permissions. For example, if the subscriber does not have the needed permissions to create subaccounts, the NWSP web application has the needed logic to omit the Accounts button from the navigation bar.

JSP Pages for User-Shape Decoration

Another smaller set of JSP pages performs some user-shape decoration functions. These JSP pages reside in the \nwsp\docroot\decorator directory. The SESM developer may modify or extend these decorator JSP pages. For information on the user-shape decoration JSP pages, see the "SESM Software Concepts" section on page 3-6 and the "Decorating a User Shape" section on page 3-16.

Servlets for the SESM Controls

The architecture of an SESM web application uses the Model-View-Control (MVC) design pattern. A Cisco SESM web application like NWSP includes a set of controls that are implemented as Java servlets. The compiled classes for the control servlets are packaged in a JAR file named sesm.jar that is located in the \nwsp\docroot\Web-inf\lib\ directory. The SESM developer performs no servlet programming. For information on the MVC design pattern and the SESM control servlets, see the "SESM Architecture: An Overview" section on page 3-2.

NWSP Templates

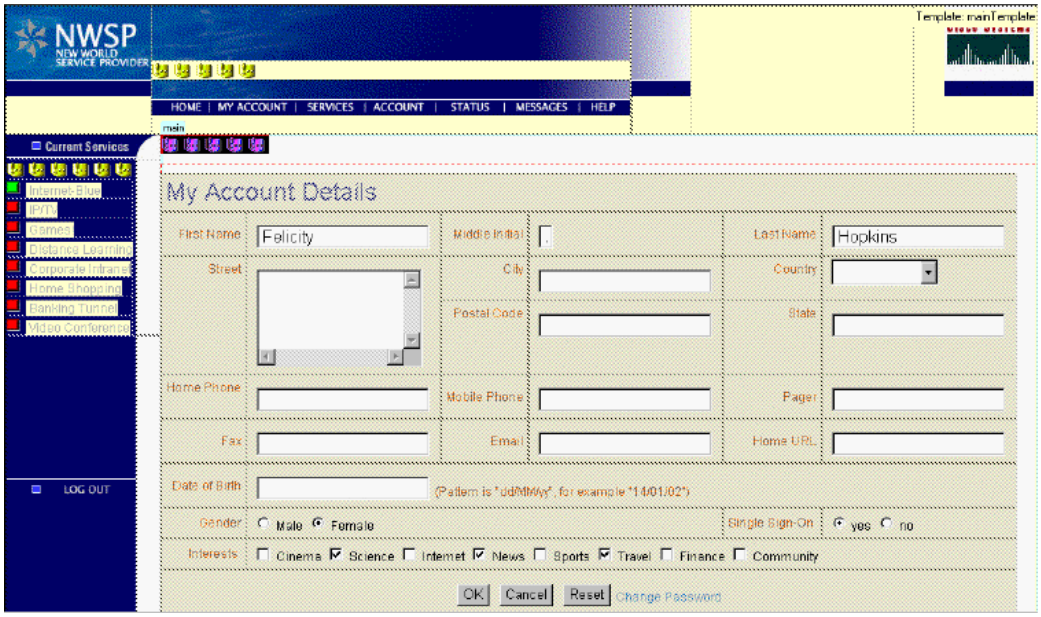
The NWSP components include two Dreamweaver templates for customizing and maintaining the JSP pages:

- mainTemplate.dwt
- bannerOnlyTemplate.dwt

Main Template

The template mainTemplate.dwt is used for NWSP pages that require a service list, navigation bar, Log Out button, and banner with brand icons. Many NWSP JSP pages, including home.jsp, use this template. Figure 4-3 shows the home.jsp page, which is derived from mainTemplate.dwt, as its appears in Dreamweaver with its table borders visible.

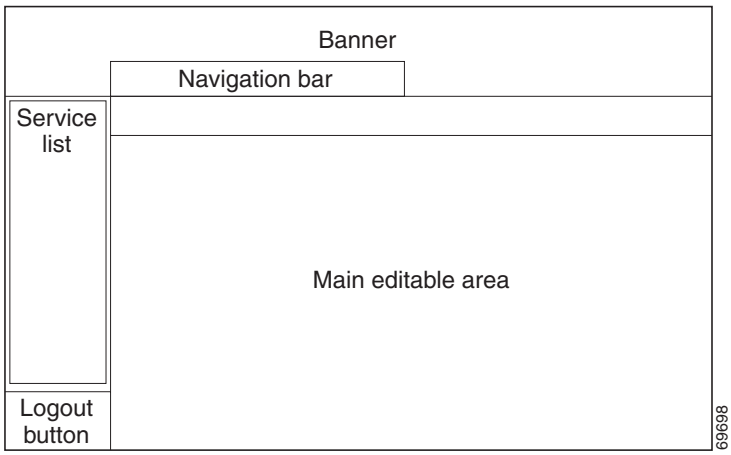
Figure 4-3 Page That Uses mainTemplate.dwt



The template mainTemplate.dwt (Figure 4-4) is a set of tables that structure the NWSP page content into the following parts:

- Banner
- Navigation bar
- Service list
- Main editable area
- Log Out button

Figure 4-4 Structure of the Template mainTemplate.dwt



The NWSP banner contains a set of images for branding.

The NWSP navigation bar is implemented with the Cisco Navigation Bar extension to Dreamweaver and a custom JavaScript, navbar.js. For information on the navigation bar, see the "NWSP Navigation Bar" section on page 4-15.

In the template, there are two editable areas: main and head. The JSP pages that are derived from mainTemplate.dwt define the content of these editable areas.

The main editable area is empty. The content of this area varies depending on the purpose of the page. For example, if the JSP page allows the subscriber to change account information, the main editable area contains the appropriate form.

The head editable area (not shown in Figure 4-4) has boilerplate code that the individual JSP pages, which are derived from the template, modify with page-specific information:

- A page title defines the text in the title bar of the browser window.
- To improve performance, the `pageOnLoad` function can preload one or more files when the JSP page is loaded. The files usually contain images for buttons used on the JSP page. For example:

```
function pageOnLoad() {
MM_preloadImages(
'<shape:file name='/images/logout_button_over.gif'/>',
'<shape:file name='/images/home_button_over.gif'/>',
'<shape:file name='/images/my_account_button_over.gif'/>',
...
);
}
```

Tip Preloading images can cause a page to load more slowly. Because the client browser caches these image files, files that have been loaded once do not need to be reloaded by the other JSP pages. The general rule is: if possible, preload a file once only.

In mainTemplate.dwt, the Log Out button is implemented with two GIF files (logout_button.gif and logout_button_over.gif). The Log Out button uses two JavaScript functions from navbar.js to swap images when an `onMouseOver` or `onMouseOut` event occurs.

NWSP Service List

The *service list* (Figure 4-5) 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. In LDAP mode, the subscriber can use an SESM web application to add or remove services from the set of subscribed services that are displayed in the service list. This feature is called *self-subscription*.

Figure 4-5 Service List with Service-Status Icons and Text

Service-Status Icons. For each service in the service list, a service-status icon indicates the state of the service. Table 4-3 lists the service-status icons that are used. The files for the images used in the service list are located in the \nwsp\docroot\images\serviceList and \nwsp\docroot\nwsp\images\serviceList directories. NWSP uses the files in the latter directory when the value of the `brand` dimension is nwsp.

Table 4-3 Service-Status Icons

Icon (color)	Description	File
(green)	Indicates that the subscriber is connected to the service.	serviceOn.gif
(red)	Indicates that the subscriber is not connected to the service.	serviceOff.gif
(red and green)	Indicates that the service connection was lost.	serviceLost.gif

For each service in the service list, the associated service-status icon has these corresponding elements:

- Service page URL
- Traffic-light image
- Alternative text
- Text for the service

As an example of how an SESM web application determines the preceding elements, consider this code from /docroot/pages/serviceListService.jsp.

```
<%@ taglib uri="http://www.cisco.com/taglibs/localization" prefix="l10n" %>
...
<jsp:useBean id="serviceBean" class="com.cisco.sesm.webapp.control.ServiceListServiceBean"
scope="request" />
...
<a href="/service<%=serviceBean.getAction()%>/home?service=<%=serviceBean.getName()%>">
<img src="<shape:file
name='<%="/images/serviceList/service"+serviceBean.getStatus()+".gif"%>'/>"
alt="<l10n:format object='<%= serviceBean.getActionDesc() %>' />" border="0">
<l10n:format object='<%= serviceBean.getDescription() %>'>description</l10n:format>
</a>
```

Note The preceding service-list code is used for browsers that do not support HTML 4. For HTML 4, the service list is implemented with JavaScript (serviceList.js). The serviceListService.jsp page for HTML 4 constructs the service list elements in a manner similar to the manner described in this section. The serviceListService.jsp for HTML 4 is located in the /docroot/html/3/4/pages directory.

Service-Page URL. In serviceListService.jsp, the service page is the page that the service-status icon links to. The URL to the service page is defined as follows:

```
href="/service<%=serviceBean.getAction()%>/home<%=cookie%>?service=
   <%=serviceBean.getName()%>">
```

A unique `serviceBean` request attribute exists for each service. The `serviceBean.getAction` method returns the subscriber's action. The action that clicking the link will perform depends on the service's status.

- If the current status of the service is On, the action is Stop.
- If the current status of the service is Off or Lost (session lost), the action is Start.

The `serviceBean.getName` method obtains the service name, as defined in the service profile. When the subscriber's action and the service name are combined, the result is one of the following service-page URLs:

```
"/serviceStop?service=service_name"
"/serviceStart?service=service_name"
```

In the web.xml file for NWSP, the URLs for `/serviceStop` and `/serviceStart` are mapped, respectively, to the `ServiceStopControl` and `ServiceStartControl` servlets, which stop and start the service (*service_name*) specified in the URL's query string.

Service-Status Icon. The URL for the service-status icon (Table 4-3) is constructed from the current status of the service:

```
<img src=
"<shape:file name='<%="/images/serviceList/service"+serviceBean.getStatus()+".gif"%>'/>"
```

The `serviceBean.getStatus` method returns the current status of the service: On, Off, or Lost. When the status is added to the directory and file information, the result is one of the following URLs:

```
"/images/serviceList/serviceOn.gif"
"/images/serviceList/serviceOff.gif"
"/images/serviceList/serviceLost.gif"
```

The `file` tag from the Shape tag library gets the actual URI of the GIF file from the virtual file name given in its `name` attribute. For example, if the user shape included a `brand` dimension with a value of `nwsp`, the actual URI for `serviceOn.gif` might be:

```
"/nwsp/images/serviceList/serviceOn.gif"
```

For information on the `file` tag, see the "file Tag" section on page A-5.

Alternative Text. A text alternative to the image is obtained with the `serviceBean.getActionDesc` method:

```
alt="<l10n:format object='<%= serviceBean.getActionDesc() %>' />"
```

The `serviceBean.getActionDesc` method returns the action description for the current status of the service. When the subscriber passes the pointer over the hyperlink for the service, the browser displays the action description.

The action descriptions are internationalized resources of type `I18nObject` that have entries in the NWSP properties files (for example, messages.properties). Table 4-4 shows the keys and values in the NWSP messages.properties file for the three possible actions.

Table 4-4 Action Descriptions from messages.properties

Service Action	Action Description key	Action Description value (English)
Stop	serviceStopDesc	Stop
Start	serviceStartDesc	Start

The value for `alt` uses the `format` tag and its `object` attribute from the Localization tag library to localize the action description. The localization extracts a value for the action description's key from a shape-specific NWSP properties file.

Text for the Service. The text that is part of the link for the service is obtained with the `serviceBean.getDescription` method:

```
<l10n:format object='<%= serviceBean.getDescription() %>'>description</l10n:format>
```

The `serviceBean.getDescription` method returns the service description, if a description is defined in the service profile. If the service profile has no description, `getDescription` returns a service name. In either case, the `getDescription` method returns a string that the method has internationalized using the value found for a service-specific key in the NWSP messages.properties files. The key is constructed from two concatenated strings as follows:

```
"service_name"+"description_in_profile"
```

When a service description is not defined in the service profile, `"description_in_profile"` is an empty string.

As shown in the code, the JSP page uses the Localization tag library's `format` tag and its `object` attribute to localize the string that `getDescription` returns. The localization extracts a value for the key for the service description from a shape-specific NWSP properties file. For information on the use of the `format` tag, see the "format Tag" section on page 5-9.

NWSP Navigation Bar

In the NWSP web application, the Dreamweaver-created navigation bar (Figure 4-6) that appears below the banner consists of a set of buttons whose display changes based on the actions of the user. For example, one image for a button in a navigation bar is used when the pointer is rolled over the button, and another image for the button is used when the button is clicked.

Figure 4-6 Navigation Bar with Buttons for LDAP Mode

Figure 4-6 shows the NWSP navigation bar with the buttons for the dynamic update features: My Account, My Services, and Sub-Accounts. These buttons are supported only when the dynamic update functionality associated with LDAP mode is available. The Home, Status, Messages, Settings, and Help buttons are displayed in both RADIUS and LDAP modes.

In LDAP mode, the set of buttons that appear in the NWSP navigation bar varies depending on the user's permissions. Various SESM features require that subscriber have appropriate permissions. For example, the ability to create subaccounts or change account information such as a password require that the subscriber have the required permissions. The NWSP web application has the required logic to determine the permissions that were granted to the subscriber and to display the corresponding set of navigation bar buttons.

The NWSP navigation bar was created with the Cisco Navigation Bar extension to Dreamweaver. This extension changes the behavior of the Dreamweaver navigation-bar tool and uses a custom script to provide some special functionality in the NWSP navigation bar. For information on using the Cisco Navigation Bar extension, see Appendix C, "Using the Cisco Navigation Bar Extension."

Note Because the NWSP navigation bar has conditional statements that display certain buttons based on subscriber permissions, you cannot use the Dreamweaver navigation-bar tool to modify the NWSP navigation bar. The HTML for the NWSP navigation bar is located in the file /nwsp/docroot/pages/navbar.jsp. *If you need to change the NWSP navigation-bar coding, you must modify it manually with an HTML code editor or a text editor.*

Each button in the NWSP navigation bar uses three images for three button states. For example, the Home button uses these images:

- home_button.gif for the up state
- home_button_over.gif for the over state
- home_button_down.jsp for the down state

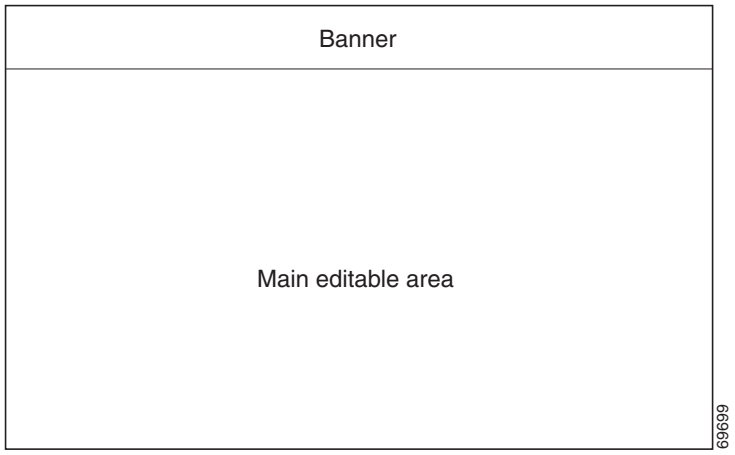
In a production SESM web application that uses the NWSP components, the developer provides any customized or localized images for each button in the navigation bar. In NWSP, GIF and JPEG files for the navigation-bar buttons are located in the \nwsp\docroot\images directory.

In the NWSP sample application, the PNG files for the buttons in the navigation bar exist in the \nwsp\docroot\assets\buttons directory. The easiest way to modify a button is to open the button's PNG file, change the image or its text, export the image to GIF or JPEG, and save the PNG file. A developer can use Fireworks or any graphics tool to modify the button images.

Banner-Only Template

The bannerOnlyTemplate.dwt file (Figure 4-7) is used for NWSP pages that do not require a service list or navigation buttons but that do require a banner with brand icons.

Figure 4-7 Structure of the Template bannerOnlyTemplate.dwt



Most JSP pages that use the template `bannerOnlyTemplate.dwt` contain message or help text: help.jsp, and message.jsp. Other JSP pages that use the template have a simple form or button or both: accountLogon.jsp, accountLogon3Key, and accountLogoff.jsp.

PDA Web Application

The sample PDA web application provides a subset of the SESM functionality for subscribers who are using a personal digital assistant (PDA) or other handheld devices. This application demonstrates how SESM might be used on a handheld for service selection. The PDA web application also shows how to provide a customized look and feel based on a brand.

The files for the PDA web application are located in the *install_dir*\pda directory. For Demo mode, the Merit RADIUS file pdademo.txt file is located in the \install_dir\pda\config directory. To determine the the subscriber and service profiles that are used for PDA application in Demo mode, see pdademo.txt.

Branding Based on User Group

The PDA web application is an example of how an SESM web application can create a look and feel customized for a specific subscriber based on a brand. With the PDA application, the user group to which a subscriber belongs determines the brand. In the sample PDA web application, there are three user groups: gold, silver, and bronze. The user `golduser` belongs to the `gold` user group, `silveruser` belongs to the `silver` user group, and `bronzeuser` belongs to the `bronze` user group. The appearance of the PDA web application is different for each user group. For example, the navigation buttons for the `gold` user group are gold in color, for the `silver` user group they are silver in color, and so on.

PDA User Interface

The PDA user interface provides a web portal for network services as well as a suite of tools such as email, text messaging, and an address organizer. The subscriber uses the web portal for selecting services and logging on to services and for using the tools. Figure 4-8 shows the home page from the PDA web application's user interface.

Figure 4-8 PDA Home Page

Designed for the small screen of a handheld device, the home page is organized into three parts:

- **Tools bar—**Buttons for the email, text messaging, and address organizer tools.
- **Service list**—A set of service names that the subscriber uses to log on to, start, and stop subscribed services.
- **Home, Logout, and Help Buttons**—Buttons that link to pages where the subscriber can log off from the SESM session or view Help information.

Note The tools linked to by the Tools bar buttons are intended for Demo mode only and are not fully functional.

PDA Functionality

The SESM-related functionality of the sample PDA web application allows the subscriber to:

- Log on to and log off from an SESM session
- Connect to or disconnect from services that are subscribed
- Log on to a service
- View help text

The service list in the PDA web application includes services but no service groups. Having no service groups in the service list is not an inherent limitation of the SESM technology but a conscious design choice dictated by the size of a handheld-device screen.

The PDA web application employs the SESM user-shape mechanisms to detect the subscriber's brand, set the `brand` dimension, and serve brand-specific resources. For information on the user-shape mechanisms, see the "User Shapes and User-Shape Decoration" section on page 3-7.

So that the PDA web application can be viewed on a standard-sized PC or workstation, the application assumes that the HTTP client device is always a PC or workstation with a standard browser.

WAP Web Application

The sample WAP web application provides a subset of the SESM functionality for WAP subscribers. This application demonstrates how SESM can be used for service selection on a WAP device.

The WAP web application is an example of how the view JSP pages can serve dynamic WAP content in markup language other than HTML. The JSP pages of the WAP web application are coded in WML.

Note A WAP phone or WAP phone simulator is needed to view the sample WAP web application. The freely available Nokia Mobile Internet Toolkit contains two simulators and tools for developing a web application that uses WML. For information on using a WAP phone simulator, see the "Using Device Simulators for WAP and WML" section on page 2-16.

The files for the WAP web application are located in the *install_dir*\wap directory. For Demo mode, the Merit RADIUS file wapdemo.txt file is located in the \install_dir\wap\config directory. To determine the the subscriber and service profiles that used for WAP application in Demo mode, see wapdemo.txt.

WAP User Interface

The WAP user interface provides a web portal for network services. The subscriber uses the web portal for selecting services and logging onto the service. Figure 4-9 shows the home page from the WAP web application's user interface.

Figure 4-9 WAP Home Page

Designed for the small screen and low bandwidth of a mobile phone, the home page is organized into two parts:

- **Logoff and Help Buttons**—Buttons that link to other pages where the subscriber can log off from the SESM session or view Help information.
- **Service list**—A set of service names that the subscriber uses to log on to, start, and stop subscribed services.

WAP Functionality

The sample WAP web application allows the subscriber to:

- Log on to and log off from an SESM session
- Connect to or disconnect from services that are subscribed
- Log on to a service
- View help text

The service list in the WAP web application includes services but no service groups. Having no service groups in the service list is not an inherent limitation of the SESM technology but a conscious design choice dictated by the size of a mobile-phone screen.

Captive Portal Web Solution

This section provides a brief overview of the SESM Captive Portal solution and describes how to use and modify the sample SESM web applications that are associated with the Cisco SESM Captive Portal solution.

For a comprehensive overview of the SESM captive solution and for information on configuring and deploying the Cisco SESM Captive Portal solution and TCP Redirect, see these documents on Cisco Connection Online at www.cisco.com:

- *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide.*
- *SSG Features in Release 12.2(4)B*

Note The explanations of the SESM Captive Portal solution and TCP Redirect feature in this guide assume that SESM web applications provide the Captive Portal functionality.

Captive Portal Solution Overview

The Cisco SESM Captive Portal solution works with the TCP Redirect for Services feature of the Service Selection Gateway (SSG) to provide TCP packet redirection and captive-portal application functionality. On the SSG, TCP Redirect is configured so that, in certain defined situations, TCP packets from a subscriber HTTP request are redirected by the SSG to an SESM Captive Portal web application.

The SESM Captive Portal application is a servlet that determines the reason for the redirection (for example, an unauthenticated subscriber). The SESM Captive Portal application then redirects the HTTP request to an SESM web application—the "content application"—that provides service content or message content through a JSP page.

- Service content might be a logon page for an unauthenticated subscriber, or a service subscription page for subscribing to a service.
- Message content might include a greetings page after logon, or an advertising page for new services that is displayed at a specified interval.

Captive Portal Solution Web Applications

A typical Cisco SESM Captive Portal solution consists of two types of web application:

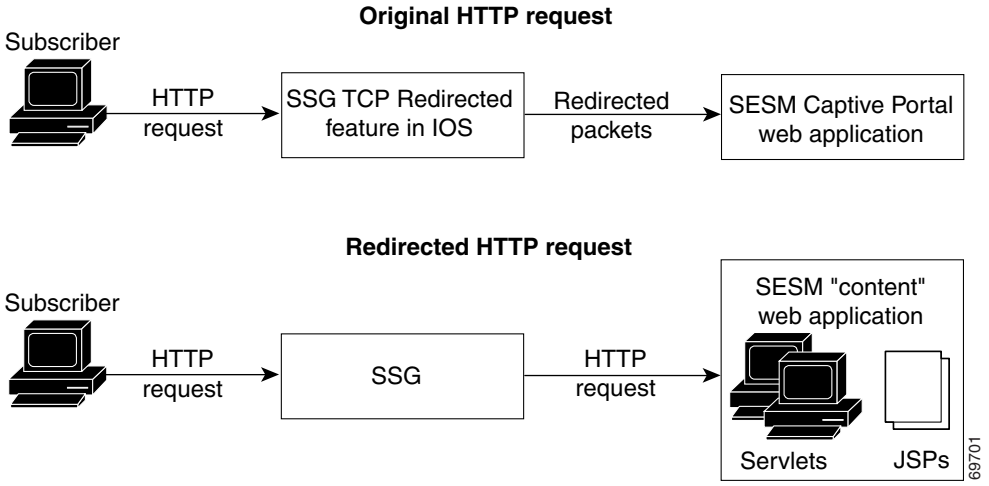
- A SESM Captive Portal web application
- One or more SESM web applications ("content applications") that provide service content or message content or both

TCP Redirect and the SESM Captive Portal application work together to redirect an HTTP request to a designated SESM content web application that provides the required service content or message content. In some cases, after the content is sent to the subscriber and displayed for a defined interval, the content web application forwards the subscriber to the URL that was originally requested. For example, after a greeting page is displayed for a number of seconds, the subscriber is forwarded to the originally requested URL.

As an example, one type of redirection that the SESM Captive Portal solution and TCP Redirect provides is unauthenticated user redirection. If a subscriber is not logged in and sends an HTTP request to one of a configurable group of TCP Redirect ports on an SSG, the following interactions occur between three components: TCP Redirect on an SSG, an SESM Captive Portal web application, and an SESM web application, which is responsible for servicing unauthenticated subscribers.

1. **TCP Redirect on an SSG:** The subscriber's browser sends a request to SSG, which redirects the packets to a specified SESM Captive Portal web application. (See Original HTTP Request in Figure 4-10.)
2. **SESM Captive Portal web application:** The SESM Captive Portal web application determines that the subscriber is not authenticated and sends a redirect response to the subscriber's browser. The redirect response specifies the SESM content web application responsible for servicing unauthenticated subscribers. The SESM Captive Portal web application captures the original URL in the subscriber's original request and sends the URL in the query-string of the redirect response.
3. **SESM content web application:** The subscriber's browser is redirected to the SESM content web application responsible for servicing unauthenticated users. (See Redirected HTTP Request in Figure 4-10.) The SESM content application responds to the subscriber with a logon page customized for the user's shape: device, brand, locale, and so on.
4. **SESM content web application:** After the subscriber sends a user name and credentials, the SESM content web application authenticates the subscriber and redirects the subscriber's browser to the originally requested URL. The originally requested URL might be for a home page setting or for a specific service.

Figure 4-10 TCP Redirect, Captive Portal, and Content Web Application



Captive Portal Solution Redirection Types

The SESM Captive Portal solution and TCP Redirect feature support the four redirection types listed in Table 4-5.

Table 4-5 Redirection Types

Redirection Type	Description
Unauthenticated user redirection	Handles attempted access to services by subscribers who have not yet authenticated to SESM. This type of redirection provides a logon page so the subscriber can authenticate and also redirects the originally requested URL so the user does not need to reenter the URL. In some configuration scenarios, such as with a point-to-point protocol (PPP) client with single sign-on enabled, the authentication is performed transparently to the subscriber.
Unauthorized service redirection	Handles unauthorized attempts to access a service. • If the access was rejected because the subscriber is not authenticated to the service, this feature provides the opportunity for the subscriber to authenticate. In some cases, the authentication can be transparent to the subscriber. • If the access was rejected because the subscriber is not authorized for the service, this feature provides the opportunity for the subscriber to become authorized. For example, it can access a billing server that allows the subscriber to make a payment on an overdue account. If the Cisco SESM is configured for LDAP mode, a service self-subscription page could be displayed.
Initial logon redirection	Displays advertising or other messages to the subscriber upon login, for a specified duration.
Advertising redirection	Displays advertising messages to the subscriber at timed intervals during an active SESM session.

Captive Portal Solution Configuration

The SSG and SESM configurations define the needed network and device information:

- The SSG configuration defines the TCP ports on which the SSG will redirect incoming TCP packets.
- The SSG configuration also specifies the SESM web server and ports where the SESM Captive Portal application listens for incoming requests.
- The SESM configuration defines a corresponding set of Captive Portal web application port numbers.
- The SESM configuration also indicates the URLs for the content web applications. The URLs for content applications can indicate the different web applications, or different pages within the same web application.

For information on configuring the SSG and SESM Captive Portal solution, see the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide* and *SSG Features in Release 12.2(4)B*.

Sample Captive Portal Solution Web Applications

The sample SESM Captive Portal solution that is included with the SESM software consists of three web applications:

- Captive Portal web application—This application used for all redirection types
- Service portal web application—The content application used for unauthenticated user redirection and unauthorized service redirection
- Message portal web application—Used for login redirection and advertising redirection

This section provides information on the sample SESM Captive Portal web application and the service and message portal web applications that are used in the Captive Portal solution.

Captive Portal Web Application

The Captive Portal web application that is included with the SESM software handles all redirection types. The Captive Portal web application performs the following functions:

- Determines the reason for the redirection (unauthenticated user, unauthorized service redirection, initial logon redirection, or advertising redirection).
- Captures information from the subscriber's original HTTP request.
- Redirects the HTTP request to an SESM service portal web application or message portal web application that provides the appropriate content. The HTTP redirect contains information about the subscriber's request in query-string parameters appended to the redirection URL.

The SSG and SESM configurations define the port numbers where the SESM Captive Portal application listens for incoming requests. The SESM captive-portal files configuration also provides host names or addresses, port numbers, and the URLs for the service portal and message portal web applications. The URLs for the servicing and messages content can indicate different web applications or different pages within the same web application. The SESM Captive Portal web application is preprogrammed and requires no modifications.

The parameters in Table 4-6 are defined in the captiveportal.xml file, located in the *install_dir*\captiveportal\config directory. These are the names of the parameters that SESM Captive Portal web application sends in the query-string of the redirect response to the service portal web application or message portal web application.

Table 4-6 Parameters Appended by Captive Portal Web Application

Redirection Type	Parameter Name	Description
Unauthenticated user redirection	`CPURL`	The URL in the subscriber's original HTTP request.
Unauthorized service redirection	`serviceURL`	The URL of the service that was requested in the subscriber's HTTP request.
	`service`	The name of the service that was requested in the subscriber's HTTP request.
	`username`	The user name from the subscriber profile.

Table 4-6 Parameters Appended by Captive Portal Web Application (continued)

Redirection Type	Parameter Name	Description
Initial logon redirection and Advertising redirection	CPURL	The URL in the subscriber's original HTTP request.
	CPDURATION	The message duration defined in the relevant captiveportal MBean attribute: • initialCaptivateDuration for initial logon redirection • advertisingCaptivateDuration for advertising redirection
	CPSUBSCRIBER	The user name from the subscriber profile.

The parameters in Table 4-6 are configurable in the captiveportal.xml file:

- You can change the names of the parameters that are appended to the redirected URL by modifying their names.
- You can indicate that the SESM Captive Portal web application should not append a parameter by setting the parameter name to an empty string. As an example, to omit the unauthorized service redirection parameter serviceRedirectSubscriberParam, delete service from the following:

```
<Set name="serviceRedirectServiceParam">service</Set>
```

For information on configuring the SESM Captive Portal solution, see the *Cisco Subscriber Edge Services Manager and Subscriber Policy Engine Installation and Configuration Guide.*

Content Web Applications

The content web applications provide content to the subscriber. The following basic functionality should be included in the content applications:

- For unauthenticated user redirections—The content application should present an SESM logon page so the subscriber can authenticate.
- For unauthorized service redirections—The content application should accept parameters from the Captive Portal application identifying the originally requested service and perform some appropriate action such as:
  - Coordinating with the SSG to authenticate to the service and then connect to the service.
  - Presenting subscription information if the subscriber is not subscribed. For example, in an LDAP deployment, the content web application can present a self-subscription page.
  - Presenting a payment page if the subscriber account requires additional funding.
- For initial logon redirection—The content web application should send pages containing general messages or advertising to the subscriber. Configuration parameters define how long the messages are display on either a global or individual subscriber basis. The web application should accept a parameter CPURL from the Captive Portal application identifying the subscriber's originally requested URL. Using the original URL, the application can perform another redirection when it is finished displaying messaging.

- For advertising redirection— The content application should send pages containing advertising messages to the subscriber. This application should perform the same functions as described above for the initial logon redirection, except that it sends advertising content rather than messages.

The sample service portal web application and message portal web application provide this basic functionality.

Service Portal Web Application

The sample service portal web application is the NWSP web application. NWSP provides the content application for unauthenticated user redirection and unauthorized service redirection. The NWSP web application, which includes a set of servlets and JSP pages, handles these types of redirection as follows:

- For the unauthenticated user redirection, the default Captive Portal configuration parameters cause the user to be redirected to /home in the NWSP web application. For an unauthenticated user, /home displays the SESM logon window.
- For unauthorized service redirection, the default Captive Portal configuration parameters cause the user to be redirected to /serviceRedirect, the `ServiceRedirectControl` servlet in the NWSP web application. For information on this servlet, see the Javadoc documentation for the `ServiceRedirectControl` class.

Message Portal Web Application

The sample message portal web application is the content application for initial logon redirection and advertising redirection. The message portal web application includes a servlet (`MessagePortalServlet`) and a set of JSP pages. `MessagePortalServlet` sets the value of two HTTP request attributes so that the message JSP pages can use them:

- `url`—The URL in the subscriber's HTTP request. This URL comes from the `CPURL` parameter in the query-string sent by the Captive Portal web application.
- `duration`—The duration that web application displays the message on the subscriber's browser. The duration comes from the `CPDURATION` parameter in the query-string sent by the Captive Portal web application.

For initial logon redirection and advertising redirection, `MessagePortalServlet` forwards the redirected request to the appropriate message JSP page based on the subscriber's interests as defined in the subscriber profile.

The message JSP page displays its content for `duration`, which is defined in the captiveportal.xml file. The JSP page then forwards the subscriber to the originally requested URL.

Before redirecting to the message JSP page, the sample message portal web application determines the device that the subscriber is using so that the appropriate JSP page serves device-specific content is used.

- If the subscriber's device is a WAP phone, the message web application looks for message JSP pages in the \wap\pages directory.
- If the subscriber's device is a PDA, the message web application looks for message JSP pages in the \pda\pages directory.
- If the subscriber's device is not a WAP phone or PDA, the message web application looks for message JSP pages in the \pages directory.

Beneath these directories, the JSP pages for initial logon redirection are in an \initial directory, and the JSP pages for advertising redirection are in an \advertising directory.

To determine what JSP page to use for each interest, the message portal web application uses information in the messageportal.xml file, located in the *\install_dir*\messageportal\config directory. The message portal application uses the JSP page that is defined for the *first interest* that the subscriber has selected from a list of possible interests on the My Account page. In a deployer-created message portal web application, the algorithm for determining what message or advertising to display is based on the requirements of the application.

For more information on `MessagePortalServlet`, see the Javadoc that is installed with the SESM software.

CHAPTER 5

SESM Internationalization and Localization

The Cisco SESM software includes a set of components that can be used for internationalization and localization. This chapter provides some general information on internationalization and localization. It also explains the Cisco SESM components and techniques that help a deployer internationalize and localize an SESM web application. This chapter discusses these topics:

For information on configuring a tag library, see the "Configuring a Tag Library" section on page A-1.

Internationalization and Localization

Internationalization is the process of designing an application so that it can be adapted for various languages and regions without programming changes. The term *i18n* is sometimes used as an abbreviation for internationalization because that word begins with i, ends with n and contains 18 characters in between. Text in status and error messages that varies depending on the culture needs to be internationalized. Other data that vary by culture include labels on web application buttons and fields, and the formats of dates, times, numbers, and currencies.

Localization is the process of adapting an application for a specific language or region by adding locale-specific components and text. The term *L10n* is sometimes used as an abbreviation for localization because that word begins with L, ends with n, and contains 10 characters in between. Typically, the localization process involves translating text messages to another language and, where required, providing locale-specific images.

SESM Components for Internationalization and Localization

The Cisco SESM web components include a set of Java classes and JSP tag libraries that help the deployer to internationalize and localize an SESM web application. These Java classes and techniques extend the classes and techniques that are part of the J2EE classes. The Localization tag library provides methods for setting and changing the localization context associated with the subscriber's HTTP session. For example, the Localization tag library allows a web application to change the locale and time zone of the subscriber's session.

It is not required that a Cisco SESM web application use the extended classes and techniques that are provided with the SESM software. An SESM web application can use conventional Java techniques for internationalization and localization. However, the classes and tags provided with the Cisco SESM software are specifically designed for use with a web application.

Localizing a Web Application

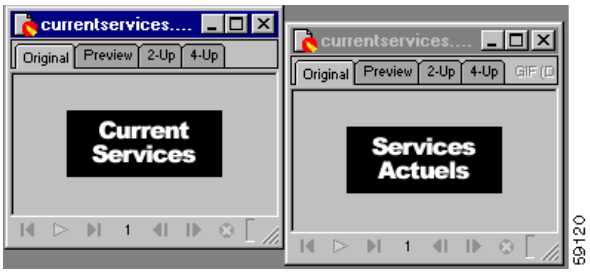
The sample SESM web applications such as NWSP use the SESM classes and techniques for internationalization and localization. The sample NWSP web application is implemented so that it dynamically detects each subscriber's language and country and generates SESM pages with resources appropriate for the language and country. For more information on these mechanisms and techniques, see the "User Shapes and User-Shape Decoration" section on page 3-7 and the "Decorating a User Shape" section on page 3-16.

The sample SESM web components, such as buttons and icons, are intended for English language subscribers. The simplest form of localization is to create an SESM web site that uses a language other than English. Creating an SESM web site for another language can be accomplished with two sets of modifications.

- Changing text, icons, and images
- Creating additional properties files

The first set of modifications involves changing SESM web application components so that they meet the needs of a language other than English. All text in icons and images must be translated into the subscriber's language. For example, the current services image in the NWSP sample contains the English language text "Current Services." If an SESM web application requires localization for the French language, the text in this image would need to be translated into the French language equivalent "Services Actuels" as shown in Figure 5-1.

Figure 5-1 Localizing currentservices.gif



The SESM web components include many of the images and icons in Portable Network Graphics (PNG) format, which is the Fireworks native format. The PNG images and icons are located in the /nwsp/docroot/assets directory. You can change the text in a PNG image using Fireworks or another image editor and then export the GIF image to the *webapp*/docroot/images directory, or a location in a sparse-tree directory structure where locale-specific images reside (for example, *webapp*/docroot/**de**/images for a set of German-language images).

The second set of modifications for localization is that message text and other items in resource bundles must be translated, and an additional properties file must be created for each new language. The NWSP web application includes the logic to determine and set the subscriber's locale so that it uses the appropriate properties file. For information on translating a properties file into another language, see the explanation at the beginning of the messages.properties file in the /nwsp/docroot/Web-inf/classes directory.

For information on resource bundles, see the "Using Resource Bundles" section on page 5-3.

Using Resource Bundles

Resource bundles contain locale-specific data that varies depending on the user's language and region, such as translatable text for status and error messages and for labels on GUI elements. A resource bundle allows a Cisco SESM deployer to separate localizable elements from the rest of the web application.

The localizable elements are stored in a set of properties files, one for each language-region combination. If an SESM web application uses resource bundles, the web application determines the subscriber's locale and then loads the appropriate resource bundle. If the subscriber switches locales, the web application can load a different resource bundle.

Resource bundles allow you to design and write an SESM web application that can be easily localized for the subscriber's language and region. An SESM web application can add additional resource bundles if a new locale is required.

The following sections provide some general information on resource bundles, properties files, and their use with a Cisco SESM web application. For detailed information on resources bundles, see the description of these classes at the Java 2 platform area of the java.sun.com web site:

- java.util.ResourceBundle
- java.util.Locale
- java.util.TimeZone
- java.text.MessageFormat

Using Properties Files

A resource bundle can be implemented with a set of one or more properties files. A *properties file* is a plain-text file that contains key-value pairs for each localizable item. For example, the English version of a sample properties file that contains message text is:

```
# English version of properties file

AMGreeting = Good Morning
PMGreeting = Good Evening
NotValid = Invalid Value
```

The French version of the properties file is:

```
# French version of properties file

AMGreeting = Bonjour
PMGreeting = Bonsoir
NotValid = Valeur Incorrecte
```

The keys and values in a properties file must be string values. A Cisco SESM web application specifies the key when it retrieves the message text from a resource bundle. The key is case-sensitive. The value associated with the key is the localized text. Properties files are not part of the Java source code. Properties files must be located in a directory specified by the `CLASSPATH` variable or in some location where the Java compiler finds web application classes. In the NWSP sample web application, the properties files (for example, messages_en.properties) reside in the \docroot\Web-inf\classes directory.

The value part of the key-value pair can include HTML markup. The value is passed straight through to the HTML page with no changes. Because the `L10nContext` class and the Localization tag library do not process special HTML characters when retrieving a value from a properties file, you can use HTML

markup in the value. Special HTML characters that appear in a value and that are not part of HTML markup must use ISO 8859-1 character encodings. For example, if you want the less-than character (<) to appear in a value as something other than HTML markup, it must be coded as &lt.

You can find information on how to write and retrieve the entries in a properties file from these sources:

- For information on the required syntax for the key-value pairs, see the description of the `java.util.Properties.load` method in the Java 2 platform area of the java.sun.com web site.
- For information on using the Localization tag library to retrieve values from a resource bundle, see the "resource Tag" section on page 5-10 and the "template Tag" section on page 5-11.

The filename of each properties file has a base name and an optional locale identifier. The optional locale identifier can include a language name, a country name, and a variant name. Elements are separated from each other by the underscore (_) character. All properties files must have the .properties extension. As an example, for the resource bundle SESMResources, Table 5-1 shows five examples of properties file names.

Table 5-1 Names of Properties Files in a Resource Bundle

Properties Filename	Description
SESMResources.properties	base name—The file is the default version.
SESMResources_en.properties	base name and language name—The file is the English version.
SESMResources_fr.properties	base name and language name—The file is the French version.
SESMResources_fr_CA.properties	base name and language name and country name—The file is the French version used for Canada.
SESMResources_fr_CA_WIN.properties	base name and language name and country name and variant name—The file is the French version used for Canada on the Windows operating system.

Properties files within the same bundle share the same base name and have the same key-value pairs. The `ResourceBundle` class associates a parent with each bundle. For example, SESMResources_fr is the parent of SESMResources_fr_CA. If the `ResourceBundle` class looks for the file SESMResources_fr_CA.properties and cannot find the file, it uses the parent file SESMResources_fr.properties or the file having the base name if it cannot find a parent file.

Note If an SESM web application uses the `L10nContext` class or the Localization tag library, the algorithm that determines the default resource bundle calls the `L10nContext.getDefault` method (not the `Locale.getDefault` method) to get the default that is associated with the SESM web application. For information on the benefits of using the `L10nContext` class and the Localization tag library, see the "Using the Localization Tag Library" section on page 5-5.

For detailed information on resource bundle properties files and the search algorithm used by the `ResourceBundle` class, see the class description in the Java 2 platform area of the java.sun.com web site.

Using the Localization Tag Library

The Cisco SESM software includes a Localization tag library that helps reduce the complexity of localizing an SESM web application. The Localization tag library uses a special SESM class (`L10nContext`) that improves upon the standard Java locale-related classes for use in a web application. The Localization tag library includes these tags:

- `context`—Sets the characteristics of the current localization context (`L10nContext`).
- `locale`, `timeZone`, and `language`—Get a string describing the specified characteristic of the current localization context.
- `country`—Gets a string for a country specified by the attribute used with the tag.
- `format`—Converts currency, number, date, and time values according the formatting conventions associated with the Java class `java.text.MessageFormat` and the current `L10nContext` localization context. Also, formats an internationalized object
- `resource`—Obtains a resource from the resource bundle specified for the current localization context and for a specified key.
- `template`—Replaces tokens in a template with parameter values. The `template` tag can be used with the `resource` tag for token replacement in a resource.

The Localization tag library is specifically designed for web application localization. The standard Java `Locale.getDefault` method gets the current value of the default locale for this instance of the Java Virtual Machine (JVM). This default locale is shared across all applications running on that JVM.

In contrast, if the `L10nContext` class and the Localization tag library is used, the class and tag library get the current value of the default locale for the class loader. If each application running on the JVM has its own class loader, then each application has its own localization-context object created using the Cisco SESM `L10nContext` class. Thus, the benefit to using the Localization tag library and the SESM `L10nContext` class is that another application running on the JVM can change the default locale associated with the `Locale` object, but it cannot change a Cisco SESM web application's default `L10nContext` object.

context Tag

The `context` tag can be used to create a scripting variable of the type `L10nContext`, specify the localization context explicitly, and set the characteristics of the current localization context (`L10nContext`). A localization context combines a locale, time zone, resource bundle base name, preferred locales, otherwise locale, and scope.

When a web application uses a `context` tag, the tag implicitly declares a localization context. The `context` tag's current localization context inherits values for locale, time zone, and resource bundle base name from the first of the following localization contexts that exists:

1. A parent `context` tag.
2. If no parent `context` tag is used, the values come from a localization context stored in an `L10nContext` object. The `context` tag software searches for a localization context having (in this order) page, request, session, or application scope. It uses the first context found.
3. If none of the preceding exist, the values come from the default localization context, which it obtains with the `L10nContext.getDefault` method.

A web application can override the current localization context's inherited values with the `context` tag attributes such as `locale`, `preferredLocales`, `timeZone`, and `resourceBundleName`. For example, a web application can set the locale of a user's localization context to Germany for the duration of an HTTP session as follows:

```
<l10n:context locale = "<%= Locale.GERMANY%>" scope = PageContext.SESSION_SCOPE />
```

A localization context object can exist for each of four scopes: page, request, session, and application. As shown in the preceding example, the `scope` attribute defines the scope of a localization context and can be specified with any other `context` tag attribute.

The current localization context, including all its characteristics, can be specified by setting the context using the `context` attribute and an existing `L10nContext` object. For example:

```
<l10n:context context = "<%= someL10nContextObject %>" />
```

If a tag in the Localization tag library is used *inside* the tag body of a `context` tag, the tag's functionality reflects the localization context. In the following example, the currency amount is formatted according to German conventions (`99,99 DM`) because the formatting occurs within the `context` tag body where the localization context's locale is Germany.

```
<% double amount = 99.99; %>

<!-- Set the locale of the current L10nContext localization context -->
<l10n:context locale = "<%= Locale.GERMANY%>" >
Amount formatted as currency:  <l10n:format currency="<%= amount %>" /> <BR>
</l10n:context>
```

If a tag in the Localization tag library is used *outside* the tag body of a `context` tag, the localization context is the same as having a parent context tag with no attributes set.

Table 5-2 lists the attributes of the context tag.

Table 5-2 Context Tag Attributes

Attribute	Description	Required	Runtime Expression
`variable`	Declares a variable that has the type `L10nContext` and the specified name. The named variable can be used as a scripting variable within the tag body. The value assigned to `variable` is the declared `L10nContext` object's name. See the example following this table.	No	No

`context`	Specifies the current localization context explicitly. For example:
	`<l10n:context`
	`context = "<%= someL10nContextObject %>" />`
		No	Yes

`resourceBundleName`	Specifies the resource bundle base name.	No	Yes

`locale`	Specifies the locale of the current localization context. For example:
	`<l10n:context locale = "<%=`
	`Locale.GERMANY%>"/>`
		No	Yes

`timeZone`	Specifies the time zone of the current localization context.	No	Yes

Table 5-2 Context Tag Attributes (continued)

Attribute	Description	Required	Runtime Expression
`preferredLocales`	Specifies the preferred locales of the current localization context. This attribute requires that an otherwise locale be specified with the `otherwise` attribute.	No	Yes
`otherwise`	Specifies the locale to use if no resource bundle can be found for any of the preferred locales. This attribute requires that a preferred locale be specified with the `preferred` attribute.	No	Yes
`scope`	Specifies the scope of the current localization context. Allowed values for scope are: • `PageContext.APPLICATION_SCOPE` • `PageContext.SESSION_SCOPE` • `PageContext.REQUEST_SCOPE` • `PageContext.PAGE_SCOPE` For the meaning of each scope, see the documentation for the Java `PageContext` class in the Java 2 platform area of the java.sun.com web site.	No	Yes

The following example shows how to declare and use the scripting variable that is created with the `context` tag's `variable` attribute. In the example, the scripting variable is used to access the `getLocale` method of the `L10nContext` class.

```
<%@ taglib uri="http://www.cisco.com/taglibs/localization" prefix="l10n" %>
...
<%-- Set the locale and encoding of the response --%>
<%-- to the current locale of the L10nContext.   --%>
<l10n:context variable="l10nContext">
<%
response.setLocale(l10nContext.getLocale());
Log.debug("decorateResponse.jspi, locale=", response.getLocale());
%>
</l10n:context>
```

locale, timeZone, and language Tags

The `locale`, `timeZone`, and `language` tags get a text description of the specified characteristic for the current localization context (`L10nContext`):

- `locale`—Gets a text description of the locale.
- `timeZone`—Gets a text description of the time zone.
- `language`—Gets a text description of the language.

The following example uses the `locale` and `timeZone` tags to obtain a text description for the locale and time zone of the current localization context.

```
<%@ taglib uri="http://www.cisco.com/taglibs/localization" prefix="l10n" %>
...
The current locale is: <l10n:locale /> <BR>
The current timeZone is: <l10n:timeZone /> <BR>
```

The generated HTML page displays the following:

```
The current locale is: English (United States)
The current timeZone is: America/New_York
```

country Tag

The `country` tag gets a text description for a country. The attribute supplied with the `country` tag specifies the country. If no attribute is used, the description is for the country of the current localization context (`L10nContext`). The text description is always in the language of the current localization context. Table 5-3 lists the attributes of the `country` tag.

Table 5-3 Country Tag Attributes

Attribute	Description	Required	Runtime Expression
`code`	Gets the text description for the country specified by a two-character ISO 3166 country code, such as “JP” for Japan. For a list of country codes, see: http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html	No	Yes
`locale`	Gets the text description for the country of the specified locale.	No	Yes
`context`	Gets the text description for the country of the specified localization context.	No	Yes

The following example uses the `country` tag and its various attributes to obtain text descriptions for some countries.

```
<%@ taglib uri="http://www.cisco.com/taglibs/localization" prefix="l10n" %>
...
<!-- No attribute specified -->
The country of the current localization context is: <l10n:country /> <BR>

<!-- Country code specified -->
The country associated with the country code "CH" is: <l10n:country code = "CH" /><BR>
<BR>

<!-- For the swissFrench object, set language to French and the country to Switzerland -->
<% Locale swissFrench = new Locale("fr", "CH"); %>

<!-- Set the locale of the current L10nContext localization context -->
<l10n:context locale = "<%= swissFrench %>" >

After changing the locale of the current localization context, <BR>
the country of the localization context <BR>
(in the language of that localization context) is: <l10n:country /> <BR>

</l10n:context>

<BR>

<!-- Set the locale of the current L10nContext localization context -->
<l10n:context locale = "<%= Locale.GERMANY%>" >

After changing the locale of the current localization context, <BR>
the country of the localization context <BR>
(in the language of that localization context) is: <l10n:country /> <BR>
```

```
</l10n:context>
```

The generated HTML page displays the following:

```
The country of the current localization context is: United States
The country associated with the country code "CH" is: Switzerland

After changing the locale of the current localization context,
the country of the localization context
(in the language of that localization context) is: Suisse

After changing the locale of the current localization context,
the country of the localization context
(in the language of that localization context) is: Deutschland
```

format Tag

The `format` tag can be used to convert currency, number, date, time, or internationalized object values into text according to the formatting conventions associated with the Java class `java.text.MessageFormat` and the current `L10nContext` localization context.

Table 5-4 lists the attributes of the `format` tag.

Table 5-4 Format Tag Attributes

Attribute	Description	Required	Runtime Expression
`currency`	Specifies a `Number`, `double`, or `long` value to be converted into a currency.	No	Yes
`number`	Specifies a `Number`, `double`, or `long` value to be converted into a number.	No	Yes
`date`	Specifies a `Date` value to be converted into a date.	No	Yes
`time`	Specifies a `Date` value to be converted into a time.	No	Yes
`dateTime`	Specifies a `Date` value to be converted using date-time format.	No	Yes
`shortDate`	Specifies a `Date` value to be converted using short date format.	No	Yes
`shortTime`	Specifies a `Date` value to be converted using short time format.	No	Yes
`shortDateTime`	Specifies a `Date` value to be converted using short date-time format.	No	Yes
`mediumDate`	Specifies a `Date` value to be converted using medium date format.	No	Yes
`mediumTime`	Specifies a `Date` value to be converted using medium time format.	No	Yes
`mediumDateTime`	Specifies a `Date` value to be converted using medium date-time format.	No	Yes
`longDate`	Specifies a `Date` value to be converted using long date format.	No	Yes

Table 5-4 Format Tag Attributes (continued)

Attribute	Description	Required	Runtime Expression
`longTime`	Specifies a `Date` value to be converted using long time format.	No	Yes
`longDateTime`	Specifies a `Date` value to be converted using long date-time format.	No	Yes
`fullDate`	Specifies a `Date` value to be converted using full date format.	No	Yes
`fullTime`	Specifies a `Date` value to be converted using full time format.	No	Yes
`fullDateTime`	Specifies a `Date` value to be converted using full date-time format.	No	Yes
`object`	Formats an internationalized object of type `I18nObject`.	No	Yes

The following example uses the `format` tag to format:

- A `double` value into a currency amount
- A `Date` value into a time
- A `Date` value into a long time

```
<%@ taglib uri="http://www.cisco.com/taglibs/localization" prefix="l10n" %>
...
<%
double amount = 99.99;
GregorianCalendar calendar = new GregorianCalendar();
Date curDateTime = calendar.getTime();
%>

The current locale is:  <l10n:locale /> <BR>
Amount formatted as currency:  <l10n:format currency="<%= amount %>" /> <BR>
Date value formatted as time:  <l10n:format time="<%= curDateTime %>" /> <BR>
Date value formatted as long time:  <l10n:format longTime="<%= curDateTime %>" /> <BR>
```

The generated HTML page displayed the following:

```
The current locale is: English (United States)
Amount formatted as currency: $99.99
Date value formatted as time: 7:59:18 PM
Date value formatted as long time: 7:59:18 PM EDT
```

resource Tag

The `resource` tag obtains a resource from the resource bundle of the current localization context (`L10nContext`) and for a specified key. A resource obtained is the value part of the key-value pair in a properties file. Table 5-5 lists the attributes of the `resource` tag.

Table 5-5 Resource Tag Attributes

Attribute	Description	Required	Runtime Expression
key	Specifies the key for which to obtain a resource.	Yes	Yes

The following example uses the `resource` tag to obtain the resource associated with the key `PleaseAuthenticate` from the messages.properties resource bundle of the NWSP sample web application.

```
<%@ taglib uri="http://www.cisco.com/taglibs/localization" prefix="l10n" %>
...
<l10n:context resourceBundleName="messages.properties" />

The resource for the key "PleaseAuthenticate" is: <l10n:resource key="PleaseAuthenticate">
Text in the tag body appears in Dreamweaver but not in the generated HTML.
</l10n:resource> <BR>
```

The generated HTML page displays the following:

```
The resource for the key "PleaseAuthenticate" is: Please log in
```

template Tag

The `template` tag can be used with the `resource` tag for token replacement in a resource. For a template (a message format) that appears in its tag body, the `template` tag replaces tokens in the template with the values specified with the `param` or `params` attributes. The syntax that you use for a token is specified by the class `java.text.MessageFormat`. The `template` tag formats locale-sensitive information such as dates, messages, and numbers using the conventions of the current localization context (`L10nContext`).

Table 5-6 lists the attributes of the `template` tag.

Table 5-6 Template Tag Attributes

Attribute	Description	Required	Runtime Expression
param	Specifies a single parameter value to substitute for a token. The `param` attribute is used when an array of parameter values is not required. The type of the parameter can be `Object` or any of the primitive types.	No	Yes
params	Specifies an array of type `Object[]` containing parameter values to substitute for one or more tokens.	No	Yes

In the following example, tokens in a message format in the `template` tag body are replaced by the values specified in `nameAndAge`:

```
<%! Object[] nameAndAge = new Object[] {"John", new Long(5)}; %>
<l10n:template params = "<%= nameAndAge %>" >
My name is {0}, I am {1,number,integer} years old.
</l10n:template >
```

The text in braces (`{ }`) is a token. The `template` tag replaces tokens with the values specified with the `params` attribute. In the preceding example, token `{0}` is replaced by element 0 of the `nameAndAge` array, and token `{1, number, integer}` is replaced by element 1 of the `nameAndAge` array. The generated HTML page displays the following:

```
My name is John, I am 5 years old.
```

The `template` tag can be used with the `resource` tag to get a resource from a resource bundle and to replace tokens in the resource with specified parameter values. In a properties file, the value part of a key-value pair is a template, a message format, that can contain one or more tokens. As an example, assume the following key-value pair in a properties file:

```
message=Error is {0} ({1,number,integer}).
```

The following `template` tag uses the `resource` tag in its body to retrieve the value from and replace tokens in the preceding properties file entry:

```
<%! Object[] errorInfo = new Object[] {"Not Found", new Long(403) };  %>

<l10n:template params = "<%= errorInfo %>" >
<l10n:resource key = "message" />
</l10n:template>
```

For the preceding example, the generated HTML page displays the following:

```
Error is Not Found (403).
```

Setting a Default Localization Context

You can use the initialization parameters for the `L10nContextDecorator` servlet to set the values of the web application default localization context. The `L10nContextDecorator` servlet creates an `L10nContext` object and adds it as an attribute having session scope. The initialization parameters for the servlet are defined in the web.xml file of each SESM web application. The value of a localization context (`L10nContext`) is determined as follows:

1. The values for the localization context come from the preferred locale of the subscriber HTTP client machine if the SESM web application supports the preferred locale. For example, a subscriber can set the preferred locale for a Windows client machine with the Regional Settings tool in Control Panel.
2. If the SESM web application does not support the preferred locale, the values for the localization context come from the web application default localization context, which you can specify with the `L10nContextDecorator` initialization parameters.

Table 5-7 lists the `L10nContextDecorator` initialization parameters that you can specify in the web.xml file for an SESM web application.

Table 5-7 L10nContextDecorator Default Initialization Parameters

Parameter	Description
defaultResourceBundle	A base name for a resource bundle properties file. For example: SESMResources.
defaultLanguage	A language code (for example, it). These codes are the lowercase, two-letter codes defined in ISO-639. You can find a full list of these codes at: http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt
defaultCountry	A country code (for example, IT). These codes are the uppercase, two-letter codes defined in ISO-3166. You can find a full list of these codes at: http://www.chemie.fu-berlin.de/diverse/doc/ISO_3166.html
defaultVariant	A variant code (for example, EURO). Variant codes are vendor and browser specific.
defaultTimeZone	A time zone ID (for example, Europe/Stockholm) or the value default. • For information on time-zone IDs, see the description of the TimeZone class in the Java 2 platform area of the java.com.sun site. • For information on the value default, see the explanation that follows this table.

When you use the value default for the defaultTimeZone parameter, a time-zone map associates a time zone with a specific locale. The L10nContext class uses the time-zone mapping for the web application default localization context. If the subscriber subsequently changes the preferred locale, the L10nContext software selects a new time zone that matches the new locale.

Two L10nContextDecorator servlet initialization parameters are used to define a time-zone map.

- timeZoneMapCountries is a set of country codes that defines the countries that are mapped to a time zone.
- timeZoneMapTimeZones is a set of time zone IDs that defines the time zones that are associated with the countries given in the timeZoneMapCountries parameter.

For both of these parameters, you can separate the values by whitespace or commas.

The following example shows how to use the timeZoneMapCountries and timeZoneMapTimeZones parameters.

```
<servlet>
    <servlet-name>L10nContext</servlet-name>
    <servlet-class>com.cisco.sesm.navigator.L10nContextDecorator</servlet-class>
    <load-on-startup>1</load-on-startup>

    <!-- Specify associations from Country -> TimeZone. -->
    <init-param>
    <param-name>timeZoneMapCountries</param-name>
    <param-value>AU,NZ,SE,PT</param-value>
    </init-param>
    <init-param>
    <param-name>timeZoneMapTimeZones</param-name>
    <param-value>
    Australia/Sydney
    Pacific/Auckland
    Europe/Stockholm
    Europe/Lisbon
    </param-value>
    </init-param>
```

```
<!-- The value "default" indicates that a time-zone map is used -->
    <init-param>
    <param-name>defaultTimeZone</param-name>
    param-value>default</param-value>
    </init-param>
  </servlet>
```

Given the preceding time-zone map, if the locale of the current localization context (`L10nContext`) were for Australia (country code `AU`), `L10nContextDecorator` uses the corresponding time-zone ID (`Australia/Sydney`) to determine a time zone. For each country specified in `timeZoneMapCountries`, there should be a corresponding time-zone ID given in the `timeZoneMapTimeZones` parameter.

For the default localization context values used for a sample SESM web application like NWSP, see the web.xml for the application.

SESM Tag Libraries

This appendix provides information on configuring an SESM tag library and describes the SESM tag libraries other than the Localization tag library:

For information on the Localization tag library, see the "Using the Localization Tag Library" section on page 5-5.

Configuring a Tag Library

If you are using an SESM tag library in a web application, you must perform the following steps to configure the tag library.

For a sample SESM web application, you *do not* need to perform this procedure. The SESM tag libraries are already defined in the web.xml file of each SESM web application. The tag library descriptor files and the `com.cisco.sesm.contextlib.jar` file already exist in, respectively, the Web-inf and Web-inf/lib directories.

Step 1 Copy the tag library's descriptor file from *install_dir*\NWSP\docroot\Web-inf to the \Web-inf directory of your web application. Table A-1 lists the tag library descriptor files.

Table A-1 SESM Tag Library Descriptor Files

SESM Tag Library	Tag Library Descriptor File
Iterator	iterator.tld
Localization	localization.tld
Navigator	navigator.tld
Shape	shape.tld

Step 2 Copy the `com.cisco.sesm.contextlib.jar` from *install_dir*\NWSP\docroot\Web-inf\lib directory to the \Web-inf\lib directory of your web application.

Step 3 Add the appropriate `<taglib>` element to your web application deployment descriptor in \Web-inf\web.xml. You can cut and paste the `<taglib>` elements for the libraries from the web.xml file for NWSP. For example, the `<taglib>` element for the Localization tag library is:

```
<taglib>
    <taglib-uri>http://www.cisco.com/taglibs/localization</taglib-uri>
    <taglib-location>localization.tld</taglib-location>
</taglib>
```

Step 4 To use the tag libraries on a JSP page, add the necessary `taglib` directive at the top of each JSP page. For example:

```
<%@ taglib uri="http://www.cisco.com/taglibs/localization" prefix="l10n" %>
```

Iterator Tag Library

The Iterator tag library provides tags that allow a JSP page to iterate over a Java collection. The Iterator tag library implements a `java.util.Iterator` and includes these tags:

- `start`—Marks the start and end of an iteration.
- `val`—Obtains the value of the current element of an iteration loop.

start Tag

Table A-2 lists the attributes of the `start` tag. The `start` tag uses either the iteration instance given in `over`, or the `Iterator` instance given in `value`. Either the `over` or `value` attribute must be supplied.

Table A-2 Start Tag Attributes

Attribute	Description	Required	Runtime Expression
`type`	Specifies the class of the variable given in `name`.	Yes	No
`name`	Specifies a variable name for accessing the value of the current element of the iteration.	Yes	No
`over`	Provides an instance of an iteration. The `start` tag invocation must include either `over` or `value`.	No	Yes
`value`	Provides an instance of an `Iterator`. The `start` tag invocation must include either `over` or `value`.	No	Yes

The following example uses the `start` tag to iterate through a collection of messages.

```
<%@ taglib uri="http://www.cisco.com/taglibs/iterator" prefix="it" %>

<it:start over="<%=messagesBean.getMessages()%>" name="message"
type="com.cisco.sesm.i18n.I18nObject">
<tr>
<td><l10n:format shortDateTime="<%= new Date()%>">timestamp</l10n:format></td>
<td><l10n:format object="<%= message %>">message</l10n:format></td>
</tr>
</it:start>
```

In the preceding example, the `messages` variable is used to access the current value of the iteration. Each message is an internationalized object of the type `com.cisco.sesm.i18n.I18nObject`.

val Tag

Table A-3 lists the attributes of the `val` tag. When the `val` tag is specified without the `value` attribute, the tag obtains the value of the current element of an iteration loop.

Table A-3 Val Tag Attributes

Attribute	Description	Required	Runtime Expression
`value`	Specifies Java code to evaluate for each iteration through the collection. The `value` attribute is optional. When a `val` tag is specified but no `value` attribute is given, the value of this iteration is the value of the current element.	No	Yes

The following example uses the `start` and `val` tags to iterate through a collection of titles.

```
<html>
 <%@ page import="java.util.*" %>
 <%@ taglib uri="http://www.cisco.com/taglibs/iterator" prefix="it" %>

 <head>
 <title>Iterator Example</title>
 </head>
 <body>
 <%
  Vector titles = ... A vector of titles of type java.lang.String
 %>
 <table>
 <!-- Define a header row for the table -->
   <tr>
     <th>Title</th>
     <th>Description</th>
   </tr>

 <!-- Iterate over titles. The current value will be in a variable called title -->
 <it:start over="<%=titles%>" name="title" type="java.lang.String">
   <tr>
     <td><it:val>a title</it:val></td>
     <td><it:val value="<%=getDescription(title)%>">a
description</it:val></td>
   </tr>
 </it:start>

 </table>
 </body>
 </html>
```

Navigator Tag Library

The Navigator tag library and its `decorate` tag allow a JSP page to invoke a decorator if decoration is needed.

decorate Tag

The `decorate` tag invokes a decorator if decoration is needed. The `decorate` tag is an efficient way to invoke a decorator because the `decorate` tag determines whether decoration is needed by calling the decorator's `decorateIfNecessary` method. For information on decorators and `decorateIfNecessary`, see the description of the `Decorator` class in the Javadoc documentation for SESM.

The decorator specified in the tag's `name` attribute must be declared as a servlet in the web application's web.xml file. For example, the servlet name `Cookie` is declared in the following entry to be associated with the class `com.cisco.sesm.navigator.CookieDecorator`.

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

The decorator specified in the `name` attribute must be registered with the `DecoratorPool` method. If it is not registered, the `decorate` tag throws a JSP-page exception.

Table A-4 lists the attributes of the `decorate` tag.

Table A-4 decorate Tag Attributes

Attribute	Description	Required	Runtime Expression
`name`	Invokes a decorator if decoration is needed.	Yes	Yes

The following example uses the `decorate` tag to invoke the decorator having the servlet name `Cookie`.

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

The effect of the preceding `decorate` tag is to call the `CookieDecorator.decorateIfNecessary` method. Any pre-decorators or post-decorators specified for `CookieDecorator` in the web.xml file are also invoked. For information on specifying pre-decorators and post-decorators, see the "Using the preDecorate and postDecorate Parameters" section on page B-2.

Shape Tag Library

The Shape tag library provides a set of tags that translate a virtual file name into an actual file name according to the current values for the dimensions of the user shape. The JSP pages of an SESM web application use the tags of the Shape tag library when retrieving a shape-specific web resource. For information on the user shapes and finding an actual file, see the "User Shapes and User-Shape Decoration" section on page 3-7 and the "Mapping a Virtual File Name to an Actual File Name" section on page 3-33.

The Shape tag library includes these tags:

- `file`—Translates a virtual file name into an actual file name according to the current values for the dimensions of the user shape.
- `path`—Translates a virtual file name into an actual file name according to the current values for the dimensions of the user shape.

The path tag provides some functionality that is not provided by the file tag and that is useful in Dreamweaver Design View (as opposed to Dreamweaver Live Data View).

Tip Use the file tag, in most situations, rather than the path tag. The file tag does not evaluate the tag body and, therefore, is more efficient. The exception is that you should use the path tag when linking a JSP page to a style sheet. The file tag does not work for linking to a style sheet.

file Tag

The file tag 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 identifying a web resource. A Cisco SESM web application employs the user-shape mechanisms for customizing the web resources served to a subscriber. In its JSP pages, the web application uses the file tag to specify each web resource that can differ according to the user's shape.

For example, if the banner.jpg file can be different depending on the user's shape, an SESM web application uses the file tag in a JSP page when specifying that resource:

```
<img src="<shape:file name='/images/banner.jpg'/>"
```

The file tag translates the virtual file name specified by the tag's name attribute into an actual file name (a URI) based on the following:

- Values for the dimensions of the user shape. The user shape is stored in an HTTP session attribute named "shape".
- Files that reside within the SESM web application (starting at document root).

If the file tag finds the web resource within the web application, it returns the actual URI. If the file tag does not find the resource, it throws an exception. The file tag is similar to calling Shape.getPath(name, application), where application is the web application.

Note The file tag does not work when linking a JSP page to a style sheet. Use the path tag when linking a JSP page to a style sheet.

Table A-5 lists the attributes of the file tag.

Table A-5 File Tag Attributes

Attribute	Description	Required	Runtime Expression
name	Specifies the path name of the virtual file that will be translated into an actual file name (URI).	Yes	Yes

The following example uses the file tag to get the actual file name for /images/nwsp_mid_banner.jpg.

```
<%@ taglib uri="http://www.cisco.com/taglibs/shape" prefix="shape" %>
...
<td><img src="<shape:file name='/images/nwsp_mid_banner.jpg'/>" width="465"
height="53"></td>
```

In the example, the `file` tag translates the path for the virtual file `/images/nwsp_mid_banner.jpg` into an actual file name (for example, `/gold/de/images/nwsp_mid_banner.jpg`) based on the values of the dimensions of the user's shape.

path Tag

Note Use the `file` tag, in most situations, rather than the `path` tag. The `file` tag does not evaluate the tag body and, therefore, is more efficient. The exception is that, in a production SESM web application, you should use the `path` tag when linking a JSP page to a style sheet. The `file` tag does not work for linking to a style sheet.

The `path` tag 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 identifying an existing web resource. The path for the virtual file is given in an HTML tag attribute, such as `src` or `href`.

In the following example, the `path` tag evaluates its tag body and replaces the virtual file name for the style sheet specified for the `href` attribute with an actual file name:

```
<shape:path attrib="href">
   <link rel="stylesheet" href="/styles/sesm.css" type="text/css">
</shape:path>
```

For example, if the user shape is determined by the `brand` dimension, the `path` tag might locate the style sheet for a gold-brand user at `/gold/styles/sesm.css`. In this case, the preceding example generates the following:

```
<link rel="stylesheet" href="/gold/styles/sesm.css" type="text/css">
```

The `path` tag can be useful in the following situation:

- If you use Dreamweaver's Design View mode (as opposed to Live Data View) to edit a JSP page
- If the runtime value for a web resource will be determined dynamically and differ from the value specified for an HTML tag attribute such as `src`, `href`, and `background`

In Design View, the value specified for an attribute such as `src` can cause Dreamweaver to display a broken link if the file does not exist at the specified location because the actual file name (URI) is determined at runtime.

If a `file` attribute is specified, the `path` tag evaluates its tag body at runtime and replaces the specified value of the `src` attribute with the value specified in the `file` attribute. Consider the following example:

```
<shape:path file="<%=getCurrentImageName(request, name)%>"
   <img src="/images/up1.gif" width=32 height=18>
</shape:path>
```

At runtime, the preceding example produces the following virtual file name, which the `path` tag then translates into an actual file name (URI) based on the user shape.

```
<img src="value_of_getCurrentImageName_call" width=32 height=18>
```

In Design View mode, Dreamweaver uses the tag body, and the preceding example produces this result:

```
<img src="/images/up1.gif" width=32 height=18>
```

Table A-6 lists the attributes of the `path` tag.

Note If you use the `id` attribute when the body of the `path` tag has dynamic content, different `id` values are needed to reflect the possible states of the dynamic content. With dynamic content, if the `id` attribute is used with the same value for all states, the cache must be cleared in order to see a change to the dynamic content.

Table A-6 Path Tag Attributes

Attribute	Description	Required	Runtime Expression
`attrib`	Defines the HTML tag attribute (for example, `src`, `href`, or `background`) that appears in the `path` tag body and whose value will be translated into an actual file name (a URI) at runtime. If no `attrib` attribute is given, the HTML tag attribute whose value is translated is `src`.	No	Yes
`file`	Specifies a replacement path for the file that will provide the web resource. At runtime, the replacement path given for `file` is the virtual file name that the SESM software translates into an actual file name (a URI) based on the user shape.	No	Yes

`id`	Provides caching of the web resource. If the `id` attribute is not used in an invocation of the `path` tag, a new instance of the `path` tag object is created every time the JSP page with the invocation is requested. The `id` value specifies a unique identifier for the `path` tag. For example:
	`<shape:path id="UP1"`
	`<img src="/images/up1.gif" width=32 height=18>`
	`</shape:path>`
	When an `id` attribute is specified, the `path` tag reuses the results of processing the initial instance of the tag if an identical use of the `path` tag is encountered. With the `id` attribute, tag instances are considered identical if the user shape in the first tag invocation matches the user shape in the second tag invocation.
	The value given for `id` must be globally unique within the scope of the web application.
		No	Yes

`idFile`	Provides caching of the web resource. The `idFile` value specifies a static replacement path for the file that will provide the web resource. At runtime, the replacement path given for `idFile` is the virtual file name that the SESM software translates into an actual file name (a URI) based on the user shape. The `idFile` attribute can be used instead of the `id` attribute when a runtime computation is not used for the replacement path. If an `idFile` attribute is specified, the `path` tag reuses the results of processing the initial instance of the tag if an identical use of the `path` tag is encountered. With the `idFile` attribute, tag instances are considered identical if the user shape in the first tag invocation matches the user shape in the second tag invocation. The use of an `id` or `file` attribute takes precedence over an `idFile` attribute.	No	Yes
`shape`	Specifies a user `Shape` object explicitly rather than using the `Shape` stored in the `"shape"` attribute of the current HTML session. The `"shape"` attribute of the HTML session is overridden if the `shape` attribute is specified in this tag or in a previous `shape` attribute that appears on the same JSP page. The most recent definition for the tag's `shape` attribute has precedence.	No	Yes

APPENDIX B

SESM Utility Servlets Quick Reference

This appendix provides brief descriptions of the utility Java servlets that are used in an SESM web application. The SESM components include a set of utility Java servlets that the web application uses to perform a variety of common tasks, such as:

- Ensuring that a session ID is present when an HTTP client does not support cookies
- Finding a resource customized for the user shape
- Redirecting a request to an insecure (HTTP) or secure (HTTPS) version of the same request
- Ensuring that the HTTP client does or does not cache an HTTP response

Most SESM utility servlets are decorators that extend the `Decorator` abstract class and are subclasses of the `Navigator` class. The SESM utility servlets that are decorators have `Decorator` in their names.

The SESM utility servlets are preprogrammed with specific functionality. They do not require customization, but because they are decorators, you can extend a utility servlet with a post-decorator.

This appendix does not provide descriptions of two specialized sets of servlets: SESM controls and dimension decorators. You can find information on these servlets in the following locations:

- For overview information on the control servlets, see the "Controls" section on page 3-2.
- For information on the dimension decorators, see the "Decorating a User Shape" section on page 3-16.

All SESM servlet classes including those for dimension decorators and controls are documented in the Javadoc documentation that is installed with the SESM software.

Tip A few utility servlets are very helpful for testing and debugging an SESM web application: `Snoop`, `TestDimensionDecorator`, `LocaleDecorator`, and `TestUserDecorator`. For information on debugging, see the "Debugging an SESM Web Application" section on page 2-13.

Using the preDecorate and postDecorate Parameters

An SESM utility servlet that is a subclass of either `Navigator` or `Decorator` can have `preDecorate` and `postDecorate` initialization parameters:

Initialization Parameter	Description
`preDecorate`	Specifies a list of names for other decorator servlets that are invoked, in the order listed, *before* the decorator declared in the `<servlet-class>` attribute is invoked.
`postDecorate`	Specifies a list of names for other decorator servlets that are invoked, in the order listed, *after* the decorator declared in the `<servlet-class>` attribute is invoked.

Depending on whether a servlet being declared in `<servlet-class>` is a subclass of `Navigator` or `Decorator`, the behavior of the `preDecorate` and `postDecorate` initialization parameters is different:

- If a servlet named in `<servlet-class>` is a subclass of `Navigator`, each decorator in the `preDecorate` list and the `postDecorate` list is always invoked. Each decorator is called by invoking its `decorateIfNecessary` method.
- If a servlet named in `<servlet-class>` is a subclass of `Decorator` and if the decoration by this servlet is needed, each decorator in the `preDecorate` list and the `postDecorate` list is invoked by calling its `decorateIfNecessary` method.

If a pre-decorator or post-decorator throws a `ServletException`, all decoration stops. No further pre-decorators or post-decorators are invoked. The servlet declared in the `<servlet-class>` attribute is not invoked if it has not already been called. If the servlet declared in the `<servlet-class>` attribute throws a `ServletException`, no post-decorators are invoked.

In the web.xml file, you configure the `preDecorate` and `postDecorate` initialization parameters in the servlet declaration. Consider the following declaration for `MyAccountView` that specifies the `VirtualFile` servlet:

```
<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>preDecorate</param-name>
        <param-value>User, NoCache</param-value>
    </init-param>
</servlet>
```

In the preceding example, the pre-decorators `User` and `NoCache` are invoked by calling their `decorateIfNecessary` methods. With subclasses of `Navigator` (such as `VirtualFile`), the pre-decorators and post-decorators *are always invoked*. They are called by invoking their `decorateIfNecessary` methods.

In the next example, the `UserDecorator` servlet is a subclass of `Decorator`.

```
<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>postDecorate</param-name>
        <param-value>InitUser, RemovePermission, RefreshShape</param-value>
    </init-param>
  </servlet>
```

In the preceding example, if decoration by `UserDecorator` is needed, the post-decorators `InitUser`, `RemovePermission`, and `NoCache` are invoked by calling their `decorateIfNecessary` methods. With subclasses of `Decorator` (such as `UserDecorator`), the pre-decorators and post-decorators *are invoked only if decoration is needed* by the decorator being declared in `<servlet-class>` (in this example, `UserDecorator`).

For more information on the `Navigator` and `Decorator` classes, see the Javadoc documentation for these classes.

SESM Utility Servlet Quick Reference

The following descriptions briefly explain the SESM utility servlets. For more information on a servlet, see the Javadoc for the servlet class. In each description, the Servlet Mapping entry is the URL-to-servlet mapping that is defined in the NWSP web.xml file. You can add a servlet mapping for any servlet or modify an existing mapping.

Alias Servlet

Servlet Mapping: None
Class: `com.cisco.sesm.navigator.Alias`

Maps a servlet name to a servlet chain, allowing servlets to be invoked in sequence. The `Alias` servlet's initialization parameters include the following:

Initialization Parameter	Description
`to`	Specifies the URI to which `Alias` forwards the request.

In the following example, `Alias` is used with the servlet name `StatusView`:

```
<servlet>
    <servlet-name>StatusView</servlet-name>
     <servlet-class>com.cisco.sesm.navigator.Alias</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>to</param-name>
        <param-value>/user/nocache/vfile/pages/status.jsp</param-value>
    </init-param>
  </servlet>
```

When the servlet named `StatusView` is requested, the `Alias` servlet forwards the HTTP request to the servlet chain `/user/nocache/vfile/pages/status.jsp`. The `to` initialization parameter for the `Alias` servlet specifies the URI to which the request is forwarded. For information on servlet chains, see the "Servlet Chaining" section on page 3-32.

BuildVersion Servlet

Servlet Mapping: `None`
Class: `com.cisco.sesm.webapp.decorator.BuildVersionDecorator`

Discovers the build version and adds a session scope attribute named `"buildVersion"`. The build version is obtained from the configuration files and identifies which version of this software is running.

The attribute is an internationalized object (`I18nObject`). The possible values of the internationalized object include resources with keys `versionNotAvailable` and `versionError`.

CacheDecorator Servlet

Servlet Mapping: `/cache/*`
Class: `com.cisco.sesm.navigator.CacheDecorator`

Tells the HTTP client to cache the HTTP response. The response is cached by the HTTP client until the shape of the subscriber changes. The `CacheDecorator` servlet writes HTTP headers into the response indicating that the response is to be cached.

If `CacheDecorator` can determine that the HTTP response content is not required because it is already cached, the request is quickly terminated with an empty response. In this case, `CacheDecorator` does not forward to any other decorator or servlet. The HTTP client will use the response that it has already cached.

For information on preventing caching of the HTTP response, see the `NoCacheDecorator` servlet.

CookieDecorator Servlet

Servlet Mapping: None
Class: `com.cisco.sesm.navigator.CookieDecorator`

Adds an attribute named `"cookie"` to the current HTTP session. The value of the attribute represents the HTTP session ID.

Each SESM JSP page is responsible for inserting the cookie into the URL just before the place where a question mark is located at the beginning query-string. If there is no question mark, it inserts it just before the place where a question mark would be located.

Tip The SESM web developer does not have to modify the NWSP web application for browsers that do not support cookies. The needed code is already present in NWSP. If the client browser *does support* cookies, the web server automatically adds a cookie with the HTTP session ID to the HTTP headers. If the client browser *does not support* cookies, the NWSP web application has the code, where it is needed, to add a session ID to the URLs that require them. NWSP does not require any additional URL rewriting.

If HTTP client does not support cookies (for example, because the subscriber has disabled cookies), an SESM web application uses the `"cookie"` attribute to write the session ID into every URL that is returned to the client.

If `CookieDecorator` can determine that the HTTP client supports cookies, the `"cookie"` attribute is set to the empty string. The `"cookie"` attribute is never null once the web application invokes `CookieDecorator`.

If this is the first HTTP response for this HTTP session, `CookieDecorator` does not know whether the HTTP client supports cookies in the HTTP headers. In this case, the `"cookie"` attribute is set to the session ID.

The following example shows a typical use of `CookieDecorator`. (In the NWSP web.xml file, `CookieDecorator` is declared with the servlet name `Cookie`.) In the example, the JSP-page code does the following:

- Uses the `decorate` tag of the Navigator tag library to call the `CookieDecorator.decorateIfNecessary` method to decorate (if needed) the HTTP session with a `"cookie"` attribute.
- Declares the `"cookie"` attribute as a JavaBean instance so that it accessed as a scripting variable.
- Embeds the `cookie` variable into the URL as `"/myAccount<%=cookie%>"`.

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

<nav:decorate name="Cookie" />
<jsp:useBean id="cookie" class="java.lang.String" scope="session" />
...
<th><a href="/myAccount<%=cookie%>">My-Account</a></th>
```

Notice that the JSP-page code inserts `<%=cookie>` into the URL just before the place where a question mark would be located at the beginning of the query-string.

For information on the `decorate` tag, see the "decorate Tag" section on page A-4.

DecoratorPool Servlet

Servlet Mapping: `/pool/*`
Class: `com.cisco.sesm.navigator.DecoratorPool`

Maps from a decorator name to a `Decorator` instance. When a JSP page is used as a decorator, it must register itself in the `jspInit` method by calling `DecoratorPool.register(JspPage)`. For information on JSP-page decorators, see the "Creating or Customizing Dimension Decorators" section on page 3-20.

EndSessionDecorator Servlet

Servlet Mapping: `/end/*`
Class: `com.cisco.sesm.navigator.EndSessionDecorator`

Ends the HTTP session by calling the `HttpSession.invalidate` method.

InsecureDecorator Servlet

Servlet Mapping: None
Class: `com.cisco.sesm.navigator.InsecureDecorator`

If the current HTTP request uses Secure Sockets Layer (SSL) encryption, `InsecureDecorator` redirects the request to an insecure (HTTP) version of the same request.

In the redirect, the new URL is based upon the current request not the original request. The current request is different from the original request when the original request invoked a servlet that has forwarded to the current servlet. The query string in the redirect is the same as the query string in the current request.

If the `InsecureDecorator` servlet is successful in redirecting the request, it sends the HTTP client an `SC_MOVED_TEMPORARILY` (302) status code in the response. In addition, `InsecureDecorator` terminates the current request by throwing a `ResponseCompleteNotice`, which is wrapped inside a `ServletException`.

For information redirecting a request with SSL encryption, see the `SecureDecorator` servlet.

HttpSniffDecorator Servlet

Servlet Mapping: None
Class: `com.cisco.sesm.navigator.HttpSniffDecorator`

Adds an `HttpSniffBean` attribute named `"httpSniffBean"` to the HTTP session. `HttpSniffBean` contains information about the HTTP client device. It obtains the information from the HTTP headers. Table B-1 provides information on the `HttpSniffBean` properties.

Table B-1 HttpSniffBean Properties

Bean Property	Description
`clientBrowserName`	The bean sets `clientBrowserName` 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)
`clientDeviceName`	The bean sets `clientDeviceName` to the type of the device running the HTTP client. Possible values are: • `pc` (personal computer) • `pda` (personal digital assistant) • `wap` (WAP phone)
`clientOSName`	The bean sets `clientOSName` 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)

In the NWSP web.xml file, the `HttpSniffDecorator` servlet is configured with a deployer-customizable post-decorator (`httpSniff.jsp`) that provides additional "browser sniffing" capabilities. Using this additional information about the client device, `httpSniff.jsp` modifies `HttpSniffBean` properties based on the characteristics that it detects. The intention is that the service-provider developer, who has knowledge of the client devices to expect, can modify `httpSniff.jsp` to provide better information on these devices. For example, the developer could modify `httpSniff.jsp` to use third-party browser-sniffing software.

Currently, `httpSniff.jsp` uses a JavaScript probe to detect client-device characteristics and sets `HttpSniffBean` properties to the appropriate values. In addition, it sets the `clientDeviceName` property to `wap` if it detects a WAP phone simulator.

L10nContextDecorator Servlet

Servlet Mapping: `/l10n/*`
Class: `com.cisco.sesm.navigator.L10nContextDecorator`

Sets the value of the web application default localization (`L10nContext`) context. In the web.xml file, the initialization parameters for `L10nContextDecorator` define the default values for the current localization context. For information on the `L0nContextDecorator` servlet initialization parameters, see the "Setting a Default Localization Context" section on page 5-12.

LocaleDecorator Servlet

Servlet Mapping: Varies
Class: `com.cisco.sesm.navigator.LocaleDecorator`

Sets the language, country, and (if used) variant of two HTTP session attributes:

- The current localization context (`L10nContext`) attribute—`"com.cisco.aggbu.l10n.context"`
- The `locale` dimension of the user shape attribute—`"shape"`

The `LocaleDecorator` servlet's initialization parameters include the following:

Initialization Parameter	Description
`language`	Specifies the language.
`country`	Specifies the country.
`variant`	Specifies the variant (if any).

In the web.xml file, the initialization parameters in a servlet declaration for a locale decorator define the locale. In the NWSP web.xml file, the servlet declaration for `France` is:

```
<servlet>
    <servlet-name>France</servlet-name>
    <servlet-class>com.cisco.sesm.navigator.LocaleDecorator</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>language</param-name>
        <param-value>fr</param-value>
    </init-param>
    <init-param>
        <param-name>country</param-name>
        <param-value>FR</param-value>
    </init-param>
    ...
  </servlet>
```

In the web.xml file, `LocaleDecorator` is used with a servlet mapping, which indicates the locale value that will be tested. For example:

```
<servlet-mapping>
    <servlet-name>France</servlet-name>
    <url-pattern>/locale=fr/*</url-pattern>
</servlet-mapping>
```

Given the preceding servlet declaration and servlet mapping for `France`, the following URL would invoke `LocaleDecorator` and set the language and country of the current localization context (`L10nContext`) to `fr` and `FR`, respectively. It also sets the `locale` dimension of the user shape to `fr/FR`.

http://someserver:8080/locale=fr/home

NoCacheDecorator Servlet

Servlet Mapping: `/cache/*`
Class: `com.cisco.sesm.navigator.NoCacheDecorator`

Prevents caching of the HTTP response by the HTTP client. The `NoCacheDecorator` servlet writes HTTP headers into the response indicating that the response is not to be cached.

For information on caching the HTTP response, see the `CacheDecorator` servlet.

OriginalURLDecorator Servlet

Servlet Mapping: None
Class: `com.cisco.sesm.navigator.OriginalURLDecorator`

Decorates the current HTTP request with an attribute named `"originalURL"`. The attribute value is the original URL that was sent from the HTTP client to the HTTP server. The original URL is the URL of the current HTTP request before any servlet forwarding takes place.

The `"originalURL"` attribute contains the complete URL of the request, including any parameters. The parameters are included as a query string regardless of whether the method is `GET` or `POST`.

To capture the original URL for later use, a JSP page can use `OriginalURLDecorator` (whose servlet name in the web.xml file is `OriginalURL`). For example:

```
<%-- Always capture original URL at start of each request. --%>
<nav:decorate name="OriginalURL" />
<jsp:useBean id="originalURL" type="java.lang.String" scope="request"/>
...
<jsp:param name = "okPage" value = "<%=originalURL%>" />
```

PermissionDecorator Servlet

Servlet Mapping: None
Class: `class com.cisco.sesm.webapp.decorator.PermissionDecorator`

Adds a `permissionBean` to the HTTP session. The bean is added as an HTTP session attribute named "`permissionBean`". The `permissionBean` contains Boolean variables indicating whether the current subscriber has permission to perform certain Cisco SESM web portal tasks, such as creating a subaccount.

The `PermissionDecorator` servlet is invoked on any JSP page where the page needs a `permissionBean` in order to determine the permissions that the subscriber has. For example, many JSP pages in NWSP need knowledge of subscriber permissions to determine what buttons (for example, the My Account and Accounts buttons) to display in the navigation bar.

Table B-2 provides information on the `permissionBean` properties.

Table B-2 permissionBean Properties

Bean Property	Description
`isSelfManage`	The value `true` indicates that the subscriber has the permissions needed to modify account information, such names, addresses, and so on. The value `false` indicates that the subscriber does not have the needed permissions.
`isServiceSelection`	The value `true` indicates that the subscriber has the permissions needed for service selection. The value `false` indicates that the subscriber does not have the needed permissions.
`isServiceSubscription`	The value `true` indicates that the subscriber has the permissions needed for service subscription. The value `false` indicates that the subscriber does not have the needed permissions.
`isSubAccountManage`	The value `true` indicates that the subscriber has the permissions needed for creating, deleting, and managing subaccounts. The value `false` indicates that the subscriber does not have the needed permissions.

RedirectRemainder Servlet

Servlet Mapping: `/redirect/*`
Class: `com.cisco.sesm.navigator.RedirectRemainder`

Sends a redirect response to the HTTP client. The redirection is from the current URL to a new URL, where the new URL is the remainder of the current URL after the current servlet name.

For example, because `RedirectRemainder` is mapped to the URL pattern `/redirect/*` in the NWSP web.xml file, the URL `/redirect/next` redirects to `/next`.

RemoveAttributeDecorator Servlet

Servlet Mapping: None
Class: `com.cisco.sesm.navigator.RemoveAttributeDecorator`

Removes an attribute from one of the following scopes: request, session, or application. The `RemoveAttributeDecorator` servlet's initialization parameters include the following:

Initialization Parameter	Description
`name`	Specifies the name of the attribute.
`scope`	Specifies the scope of the attribute: request, session, or application.

You can use this decorator to undo the decoration of other decorators. Given the following web.xml file declaration, when `RefreshShape` is requested, `RemoveAttributeDecorator` removes the `"shape"` attribute, which has session scope.

```
<servlet>
    <servlet-name>RefreshShape</servlet-name>
    <servlet-class>
    com.cisco.sesm.navigator.RemoveAttributeDecorator
    </servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>name</param-name>
        <param-value>shape</param-value>
    </init-param>
    <init-param>
        <param-name>scope</param-name>
        <param-value>session</param-value>
    </init-param>
    <...
</servlet>
```

SecureDecorator Servlet

Servlet Mapping: None
Class: `com.cisco.sesm.navigator.SecureDecorator`

If the current HTTP request does not use Secure Sockets Layer (SSL) encryption, `SecureDecorator` redirects the request to a secure (HTTPS) version of the same request. As an example, if the request were:

http://someserver:8080/home

The `SecureDecorator` servlet redirects the request to:

https://someserver:8080/home

In the redirect, the new URL is based upon the current request not the original request. The current request is different from the original request when the original request invoked a servlet that has forwarded to the current servlet. The query string in the redirect is the same as the query string in the current request.

If the `SecureDecorator` servlet is successful in redirecting the request, the servlet sends the HTTP client an `SC_MOVED_TEMPORARILY` (302) status code in the response. In addition, `SecureDecorator` servlet terminates the current request by throwing a `ResponseCompleteNotice`, which is wrapped inside a `ServletException`.

For information on redirecting a request without using SSL encryption, see the `InsecureDecorator` servlet.

ShapeDecorator Servlet

Servlet Mapping: `/shape/*`
Class: `com.cisco.sesm.navigator.ShapeDecorator`

Creates a `Shape` object for the user and ensures that there is a `"shape"` attribute with session scope that holds the `Shape` object. The `Shape` servlet's initialization parameters include the following:

Initialization Parameter	Description
`dimensions`	Specifies a set of zero or more dimension IDs that the SESM software uses when it creates the dimensions for a user shape.

A `Shape` object encapsulates the set of characteristics that define the web resources available for a specific 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.

For information on the `ShapeDecorator` servlet, see the "Configuring User-Shape Dimensions" section on page 3-16.

Snoop Servlet

Servlet Mapping: `/snoop/*`
Class: `com.cisco.sesm.navigator.Snoop`

For testing purposes, displays information about the HTTP session and request. The content type of the response that `Snoop` sends is `"text/plain"` so that you can view the information on different types of devices (for example, HTML browsers and WAP phones).

`Snoop` is used for debugging an SESM web application and is not used in production applications.

SESMSessionDecorator Servlet

Servlet Mapping: `/session/*`
Class: `com.cisco.sesm.webapp.decorator.SESMSessionDecorator`

Adds an attribute named `"SESMSession"` to the current HTTP *request*. The value of the attribute represents the active SESM session, which has been synchronized with the SSG.

If a new `"SESMSession"` attribute is successfully created and authenticated, `initUser.jsp` executes. The `initUser.jsp` page is a deployer-customizable decorator that you can modify to perform any subscriber-related initialization tasks that need to be accomplished after it is known that the user is authenticated. For example, `initUser.jsp` could be used to set the current localization (`L10nContext`) to the locale defined in the subscriber profile.

TestDimensionDecorator Servlet

Servlet Mapping: Varies
Class: `com.cisco.sesm.navigator.TestDimensionDecorator`

For testing purposes, sets a dimension of the user shape to a specified value. The `TestDimensionDecorator` servlet's initialization parameters include the following:

Initialization Parameter	Description
`id`	Specifies the dimension ID.
`value`	Specifies a value for the dimension identified by `id`.

In the web.xml file, the initialization parameters in a servlet declaration for the test dimension decorator define the dimension ID and the value. In the NWSP web.xml file, the servlet declaration for `WAPDevice` is:

```
<servlet>
    <servlet-name>WAPDevice</servlet-name>
    <servlet-class>com.cisco.sesm.navigator.TestDimensionDecorator</servlet-class>
    <load-on-startup>1</load-on-startup>
    <init-param>
        <param-name>id</param-name>
        <param-value>device</param-value>
    </init-param>
    <init-param>
        <param-name>value</param-name>
        <param-value>wap</param-value>
    </init-param>
    ...
</servlet>
```

In the web.xml file, `TestDimensionDecorator` is used with a servlet mapping, which indicates the dimension value that will be tested. For example:

```
<servlet-mapping>
    <servlet-name>WAPDevice</servlet-name>
    <url-pattern>/device=wap/*</url-pattern>
</servlet-mapping>
```

Given the preceding servlet declaration and servlet mapping for `WAPDevice`, the following URL would invoke `TestDimensionDecorator` and set the `device` dimension to `wap`.

http://someserver:8080/device=wap/home

TestUserDecorator Servlet

Servlet Mapping: Varies
Class: `com.cisco.sesm.navigator.TestUserDecorator`

For testing purposes, authenticates a user name and password. The user names and passwords that can be authenticated with `TestUserDecorator` are those defined in the demo.txt file, which is used for demonstration mode. The `TestDimensionDecorator` servlet's initialization parameters include the following:

Initialization Parameter	Description
`username`	Specifies a user name given in the demo.txt file.
`password`	Specifies the password for the user identified in `username`.

In the web.xml file, the initialization parameters in a servlet declaration for a test user decorator define the user name and the password. In the NWSP web.xml file, the servlet declaration for `golduser` is:

```
<servlet>
    <servlet-name>golduser</servlet-name>
    <servlet-class>com.cisco.sesm.webapp.decorator.TestUserDecorator</servlet-class>
    <load-on-startup>0</load-on-startup>
    <init-param>
        <param-name>username</param-name>
        <param-value>golduser</param-value>
    </init-param>
    <init-param>
        <param-name>password</param-name>
        <param-value>cisco</param-value>
    </init-param>
...
</servlet>
```

In the web.xml file, `TestUserDecorator` is used with a servlet mapping, which indicates the user name and password value that will be tested. For example:

```
<servlet-mapping>
    <servlet-name>golduser</servlet-name>
    <url-pattern>/user=golduser/*</url-pattern>
  </servlet-mapping>
```

Given the preceding servlet declaration and servlet mapping for `golduser`, the following URL would invoke `TestUserDecorator` and try to authenticate the user name `golduser` with the password `cisco`.

http://someserver:8080/user=golduser/home

UserDecorator Servlet

Servlet Mapping: `/user/*`
Class: `com.cisco.sesm.webapp.decorator.UserDecorator`

Ensures that the user is known. `UserDecorator` adds the user name as an attribute to the HTTP session. This may require the user to authenticate. If the user is not known, the `getNewUser` method is called.

Tip Many SESM JSP pages use `UserDecorator` as a pre-decorator to ensure that the subscriber cannot view a web page until authentication has occurred.

With `UserDecorator`, decoration is necessary until the user is known and the user is authenticated. SESM/SSG authentication can be lost without notification, leaving the web application with a user name but no SESM authentication. If you want to ensure that there is an authenticated user, invoke `UserDecorator`. For example, a JSP-page view might not care what the user name is, but by invoking `UserDecorator`, it ensures that the user is authenticated and that authentication takes place.

VirtualFile Servlet

Servlet Mapping: `/vfile/*`
Class: `com.cisco.sesm.navigator.VirtualFile`

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. The `VirtualFile` servlet's initialization parameters include the following:

Initialization Parameter	Description
`vfile`	Specifies the path name for a virtual file.

You can specify the virtual file name in either of two ways:

- In the web.xml file, as the value of the `vfile` initialization parameter to `VirtualFile`
- In a JSP page of an SESM web application, as the remainder of the URL in a servlet chain

In the web.xml file, you can specify the virtual file name as the value of the `vfile` initialization parameter to `VirtualFile`. Many of the NWSP views use this method to invoke `VirtualFile`. For example:

```
<servlet>
    <servlet-name>AccountLogonView</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/accountLogon.jsp</param-value>
    </init-param>
    ...
</servlet>
```

In a JSP page of an SESM web application, the `VirtualFile` servlet can be invoked when the web application needs to find a resource that may differ according to the user shape. The virtual file name is the remainder of the URL in a servlet chain. In the following example, a URL for `/pages/help.jsp` contains a servlet chain that invokes `VirtualFile` (which is mapped to the URL `/vfile/*`).

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

In a servlet chain, `VirtualFile` (`vfile`) must be located immediately before the virtual file name for the web resource—in the preceding example, immediately before `/pages/help.jsp`. For more information on the `VirtualFile` servlet, see "Mapping a Virtual File Name to an Actual File Name" section on page 3-33.

APPENDIX C

Using the Cisco Navigation Bar Extension

This appendix explains how you install and use the Cisco Navigation Bar (NavBar) extension to Dreamweaver.

The Cisco Navigation Bar extension changes the behavior of the standard Dreamweaver navigation bar. With the extension, the current web page always displays the appropriate down-state button in the navigation bar. To accomplish this change, a JavaScript function `MM_nbIsActive(name)` determines if the passed image name should be shown in the down state.

The Cisco Navigation Bar extension is named navbar.mxp and is located in the *install_dir*\nwsp\docroot\assets directory.

Creating a Navigation Bar

Note If your Dreamweaver installation has not been extended for the Cisco Navigation Bar extension, you may need to install the extension for some of the following procedures. For installation directions, see the "Installing the Navigation Bar Extension" section on page C-4.

The manner in which you insert a navigation bar that uses the Cisco Navigation Bar extension varies depending on whether:

- It is a new JSP page.
- The JavaScript for the navigation bar is already on the JSP page itself.
- The JavaScript for the navigation bar is incorporated on the JSP page through an `include` directive specifying navbar.js.

The directions given in the following sections also apply if you want to add the navigation bar to a Dreamweaver template.

Note After you create and insert a navigation bar that uses the Cisco Navigation Bar extension and edit the JavaScript, you cannot use Dreamweaver to modify the navigation bar (by clicking Navigation Bar from the Modify menu). Attempting to modify a navigation in this manner will corrupt the JavaScript code in the navigation bar's anchor tags (<a>).

New JSP Page

If the JSP page where you will insert the navigation bar is newly created (not an existing JSP page from an SESM web application), you can add a new navigation bar that uses the Cisco Navigation Bar extension by performing the following steps:

Step 1 Insert the navigation bar in the normal manner with Dreamweaver by choosing **Interactive Images** and then **Navigation Bar** from the **Insert** menu.

Step 2 In the created JavaScript, ensure that the `MM_nbIsActive` function is defined as follows:

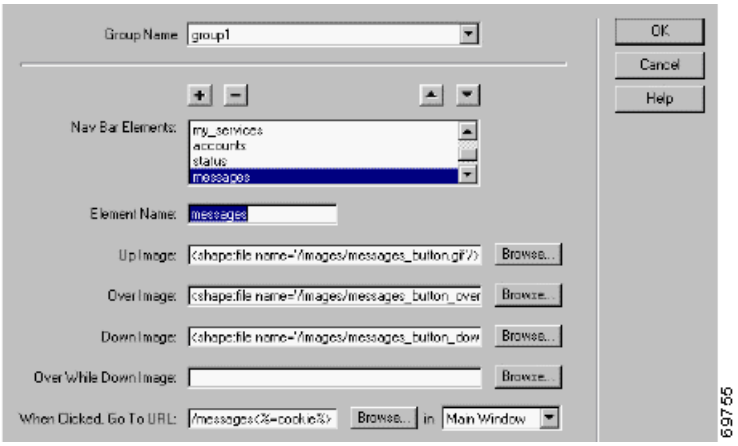
```
function MM_nbIsActive(name) { //v3.0
     return (name == '<%= selectedNavbarButton %>');
}
```

Step 3 Insert the following code above the `<head>` section of the JSP page where the navigation bar is used. The code is used by the navigation bar software to determine the button that should be in the down state on a given page.

```
<% String selectedNavbarButton = "this_element"; %>
```

In the preceding code, *`this_element`* is the name of an element within the Dreamweaver navigation bar. Each button in the navigation bar has an element name. For example, in the Dreamweaver Insert Navigation Bar dialog box shown in Figure C-1, `messages` is the element name.

Figure C-1 Navigation Bar Element Names



The string for *`this_element`* is different for each JSP page and corresponds to the element name for the page's button in the navigation bar. As an example, in the NWSP web application, the element name of the button that links to the Messages web page is `messages`. In messages.jsp, the code would be:

```
<% String selectedNavbarButton = "messages"; %>
```

JSP Page with Existing JavaScript

If an existing JSP page where you will insert the navigation bar already contains JavaScript for a navigation bar, Dreamweaver will not insert JavaScript for a new navigation bar. In this case, to add a new navigation bar that uses the Cisco Navigation Bar extension, perform the following steps:

Step 1 Copy navbar.js from the install_dir/nwsp/docroot/decorators directory into the document tree of the SESM web application. The navbar.js file contains the custom JavaScript functions that the navigation bar requires.

Step 2 In the JSP page, delete the JavaScript for the existing navigation bar.

Step 3 Insert the following code into the `<head>` section of the JSP page where the navigation bar is used. (If a JSP page is derived from a template that has the code, the code is automatically copied to the JSP page when it is created.)

```
<script language="JavaScript">
<%@ include file="/location/navbar.js" %>
</script>
```

In the preceding code, *location* is the directory where navbar.js is located within the document tree of the web application. For example: /decorators/navbar.js.

Step 4 In the navbar.js file, ensure that the `MM_nbIsActive` function is defined as follows:

```
function MM_nbIsActive(name) { //v3.0
     return (name == '<%= selectedNavbarButton %>');
}
```

Step 5 Insert the following code above the `<head>` section of the JSP page where the navigation bar appears. The navigation bar software uses the code to determine the button that should be in the down state on a given page.

```
<% String selectedNavbarButton = "this_element"; %>
```

For more information on step 5, see the "New JSP Page" section on page C-2.

JSP Page with Included JavaScript

If an existing JSP page where you will insert the navigation bar already has included navbar.js through the use of an `include` directive, Dreamweaver inserts the JavaScript for the new navigation bar. In this case, to add a new navigation bar that uses the Cisco Navigation Bar extension, perform the following steps:

Step 1 Delete the JavaScript that Dreamweaver added for the new navigation bar. (The same JavaScript code is contained in the navbar.js file.)

Step 2 Ensure that the `MM_nbIsActive` function in navbar.js is defined as follows:

```
function MM_nbIsActive(name) { //v3.0
     return (name == '<%= selectedNavbarButton %>');
}
```

Step 3 Ensure that the following code is inserted above the `<head>` section of the JSP page where the navigation bar appears. The navigation bar software uses the code to determine the button that should be in the down state on a given page.

```
<% String selectedNavbarButton = "this_element"; %>
```

For more information on step 3, see the "New JSP Page" section on page C-2.

Installing the Navigation Bar Extension

When you install the Cisco Navigation Bar extension, you will overwrite a number of Dreamweaver files that are used for creating a standard Dreamweaver navigation bar. Installing the Cisco Navigation Bar extension overwrites the following files:

- \Configuration\Behaviors\Actions\Set Nav Bar Image.js
- \Configuration\Commands\Insert Nav Bar.htm
- \Configuration\Commands\Modify Nav Bar.htm
- \Configuration\Commands\NavigationBar.htm
- \Configuration\Objects\Common\NavigationBar.js
- \Configuration\Configuration\Shared\MM\Scripts\navBar.js

The files are located below the directory where Dreamweaver or Dreamweaver UltraDev is located (for example, the C:\Program Files\Macromedia*Dreamweaver_or_UltraDev 4* directory.

Note If you want to be able to restore an installation of Dreamweaver so that a standard navigation bar can be created, make backup copies of the preceding files under a name like *filename*.ORIGINAL before you install the Cisco Navigation Bar extension.

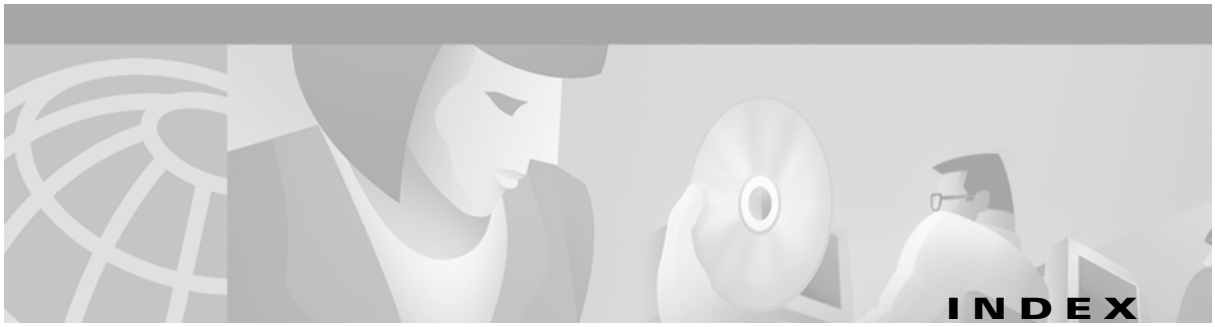
To install the Cisco Navigation Bar extension, do the following:

Step 1 If you have not already done so, install the Macromedia Extension Manager software, which can be downloaded from the Macromedia web site: www.macromedia.com.

Step 2 Copy the Cisco Navigation Bar extension package file (navbar.mxp) from the \install_dir\nwsp\docroot\assets to the Extensions directory, which is located below the directory where Dreamweaver or Dreamweaver UltraDev is located. For example:

C:\Program Files\Macromedia\Dreamweaver UltraDev 4\Extensions

Step 3 Start Macromedia Extension Manager, choose **Install Extension** from the **Extension Manager File,** and install the Cisco Navigation Bar extension (Cisco NavBar modification).



A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X