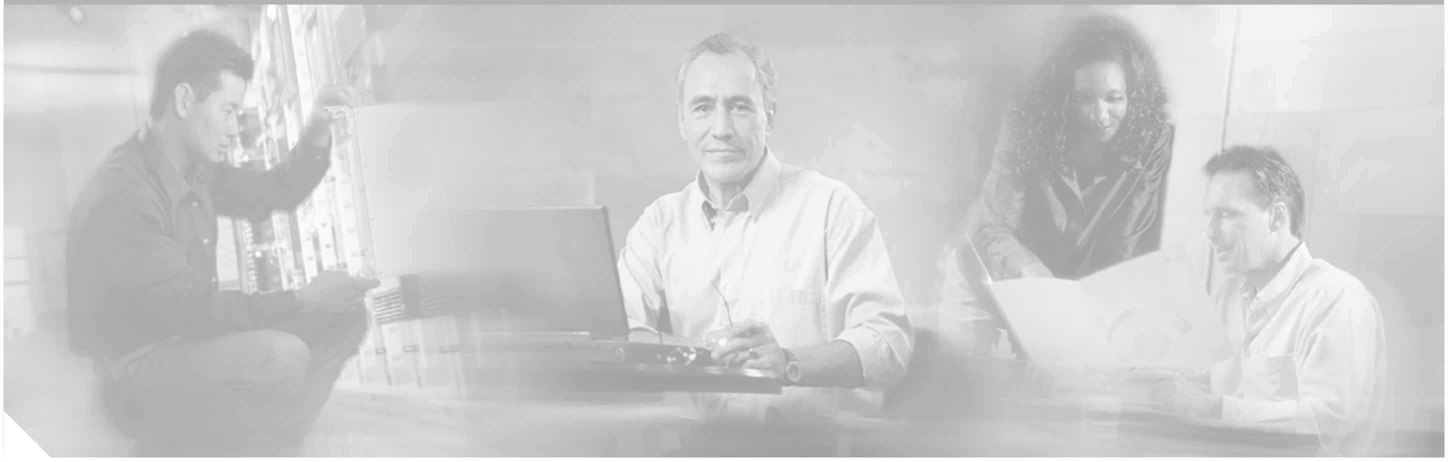
CISCO SYSTEMS



Cisco Subscriber Edge Services Manager Web Services Gateway Guide

SESM 3.3

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

Text Part Number: OL-4567-02

CONTENTS

About This Guide

This preface introduces the *Cisco Subscriber Edge Services Manager Web Services Gateway Guide*. The preface contains the following sections:

- Document Objectives
- Audience
- Document Organization
- Document Conventions
- Related Documentation
- Obtaining Documentation
- Obtaining Technical Assistance
- Obtaining Additional Publications and Information

Document Objectives

This guide describes the features and functionality of the Cisco Subscriber Edge Services Manager (Cisco SESM) Web Services Gateway (WSG).

Audience

This guide is intended for anyone who is planning to use a WSG for integration to the SESM solution.

Document Organization

This guide contains the chapters shown in the following table:

Chapter	Title	Description
Chapter 1	Using the SESM Web Services Gateway	Describes the SESM WSG and contains instructions on how to install, start, and use the WSG applications.
Chapter 2	Web Services Gateway	Provides a description of the WSG and describes the operations that are supported by the interface.

Chapter	Title	Description
Appendix A	Web Services Gateway Client	Describes how to use the client interface script that provides command line access to the WSG.
Appendix B	Web Services Gateway Interface	Contains the full details of the sa_sesm.wsdl file, which provides a description of the WSG interface.

Document Conventions

The following conventions are used in this guide:

- *Italic* font is used for parameters for which you supply a value, for emphasis, and to introduce new terms.
- **Bold** font is used for user entry and command names.
- `Computer` font is used for examples.

Note Means *reader take note*. Notes contain helpful suggestions or references to materials not contained in this guide.

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

Related Documentation

Documentation for the Cisco SESM consists of:

- *Cisco Subscriber Edge Services Manager Release Notes for Release 3.3(1)*
- *Cisco Subscriber Edge Services Manager Introduction*
- *Cisco Subscriber Edge Services Manager Installation Guide*
- *Cisco Subscriber Edge Services Manager Administration and Configuration Guide*
- *Cisco Subscriber Edge Services Manager Web Portal Guide*
- *Cisco Subscriber Edge Services Manager Profile Management Guide*
- *Cisco Subscriber Edge Services Manager Web Services Gateway Guide*
- *Cisco Subscriber Edge Services Manager Web Developer Guide*
- *Cisco Subscriber Edge Services Manager SDK Programmer Guide*
- *Cisco Subscriber Edge Services Manager Troubleshooting Guide*
- *Cisco Subscriber Edge Services Manager Release Notes for Release 3.3*

Documentation for SESM is online at:

http://www.cisco.com/univercd/cc/td/doc/solution/sesm/index.htm

Documentation for the Cisco SSG is online at:

http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122newft/122t/122t13/ssg/

Information related to configuring the SSG authentication, authorization, and accounting features is included in:

- *Cisco IOS Security Configuration Guide:*

  http://www.cisco.com/univercd/cc/td/doc/product/software/ios123/123cgcr/sec_vcg.htm
- *Cisco IOS Security Command Reference*

If you are including the Cisco Access Registrar (a RADIUS server) in your SESM deployment, see the documentation for Cisco Access Registrar (AR) online at:

http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/cnsar/index.htm

Obtaining Documentation

Cisco provides several ways to obtain documentation, technical assistance, and other technical resources. These sections explain how to obtain technical information from Cisco Systems.

Cisco.com

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

http://www.cisco.com/univercd/home/home.htm

You can access the Cisco website at this URL:

http://www.cisco.com

International Cisco websites can be accessed from this 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 may have shipped with your product. The Documentation CD-ROM is updated regularly and may be more current than printed documentation. The CD-ROM package is available as a single unit or through an annual or quarterly subscription.

Registered Cisco.com users can order a single Documentation CD-ROM (product number DOC-CONDOCCD=) through the Cisco Ordering tool:

http://www.cisco.com/en/US/partner/ordering/ordering_place_order_ordering_tool_launch.html

All users can order annual or quarterly subscriptions through the online Subscription Store:

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

Ordering Documentation

You can find instructions for ordering documentation at this URL:

http://www.cisco.com/univercd/cc/td/doc/es_inpck/pdi.htm

You can order Cisco documentation in these ways:

- Registered Cisco.com users (Cisco direct customers) can order Cisco product documentation from the Networking Products MarketPlace:

  http://www.cisco.com/en/US/partner/ordering/index.shtml

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

Documentation Feedback

You can submit comments electronically on Cisco.com. On the Cisco Documentation home page, click **Feedback** at the top of the page.

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

You can submit comments by using the response card (if present) behind the front cover of your document or by writing to the following address:

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

We appreciate your comments.

Obtaining Technical Assistance

For all customers, partners, resellers, and distributors who hold valid Cisco service contracts, the Cisco Technical Assistance Center (TAC) provides 24-hour, award-winning technical support services, online and over the phone. Cisco.com features the Cisco TAC website as an online starting point for technical assistance.

Cisco TAC Website

The Cisco TAC website (http://www.cisco.com/tac) provides online documents and tools for troubleshooting and resolving technical issues with Cisco products and technologies. The Cisco TAC website is available 24 hours a day, 365 days a year.

Accessing all the tools on the Cisco TAC website requires a Cisco.com user ID and password. If you have a valid service contract but do not have a login ID or password, register at this URL:

http://tools.cisco.com/RPF/register/register.do

Opening a TAC Case

The online TAC Case Open Tool (http://www.cisco.com/tac/caseopen) is the fastest way to open P3 and P4 cases. (Your network is minimally impaired or you require product information). After you describe your situation, the TAC Case Open Tool automatically recommends resources for an immediate solution. If your issue is not resolved using these recommendations, your case will be assigned to a Cisco TAC engineer.

For P1 or P2 cases (your production network is down or severely degraded) or if you do not have Internet access, contact Cisco TAC by telephone. Cisco TAC engineers are assigned immediately to P1 and P2 cases to help keep your business operations running smoothly.

To open a case by telephone, use one of the following numbers:

Asia-Pacific: +61 2 8446 7411 (Australia: 1 800 805 227)
EMEA: +32 2 704 55 55
USA: 1 800 553-2447

For a complete listing of Cisco TAC contacts, go to this URL:

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

TAC Case Priority Definitions

To ensure that all cases are reported in a standard format, Cisco has established case priority definitions.

Priority 1 (P1)—Your network is “down” or there is a critical impact to your business operations. You and Cisco will commit all necessary resources around the clock to resolve the situation.

Priority 2 (P2)—Operation of an existing network is severely degraded, or significant aspects of your business operation are negatively affected by inadequate performance of Cisco products. You and Cisco will commit full-time resources during normal business hours to resolve the situation.

Priority 3 (P3)—Operational performance of your network is impaired, but most business operations remain functional. You and Cisco will commit resources during normal business hours to restore service to satisfactory levels.

Priority 4 (P4)—You require information or assistance with Cisco product capabilities, installation, or configuration. There is little or no effect on your business operations.

Cisco Developer Support Program

The Developer Support Program was developed to provide formalized support for Cisco interfaces to accelerate the delivery of compatible solutions to Cisco customers. The program web site at http://www.cisco.com/go/developersupport provides a central resource point for all your development needs.

Program Benefits

- Product and document downloads
- Bug reports
- Sample scripts
- Frequently Asked Questions
- Access to Developer Support Engineers

Many of the product and document downloads are accessible with a Cisco.com guest level login. However, as a member of the program, you will get access to all the program benefits listed above to promote your development efforts. The subscription also provides the ability to open support cases using the same infrastructure and processes used by Cisco Technical Assistance Center (TAC).

Our Subscription membership is fee-based. The Developer Support Agreement, with the subscription fees and list of supported interfaces, is available on the Developer Support Web site.

Note The Cisco TAC does NOT provide support for this API/interface under standard hardware or software support agreements. All technical support for this API/interface, from initial development assistance through API troubleshooting/bugs in final production applications, is provided by Cisco Developer Support and requires a separate Developer Support contract. When opening cases, a Developer Support contract number must be provided in order to receive support.

Contacting Cisco Developer Support

You can contact Cisco Developer Support using the following:

- Email: developer-support@cisco.com
- Web: http://www.cisco.com/go/developersupport

Obtaining Additional Publications and Information

Information about Cisco products, technologies, and network solutions is available from various online and printed sources.

- *The Cisco Product Catalog* describes the networking products offered by Cisco Systems, as well as ordering and customer support services. Access the *Cisco Product Catalog* at this URL:

  http://www.cisco.com/en/US/products/products_catalog_links_launch.html

- Cisco Press publishes a wide range of networking publications. Cisco suggests these titles for new and experienced users: Internetworking Terms and Acronyms Dictionary, Internetworking Technology Handbook, Internetworking Troubleshooting Guide, and the Internetworking Design Guide. For current Cisco Press titles and other information, go to Cisco Press online at this URL:

  http://www.ciscopress.com

- Packet magazine is the Cisco quarterly publication that provides the latest networking trends, technology breakthroughs, and Cisco products and solutions to help industry professionals get the most from their networking investment. Included are networking deployment and troubleshooting tips, configuration examples, customer case studies, tutorials and training, certification information, and links to numerous in-depth online resources. You can access Packet magazine at this URL:

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

- iQ Magazine is the Cisco bimonthly publication that delivers the latest information about Internet business strategies for executives. You can access iQ Magazine at this URL:

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

- Internet Protocol Journal is a quarterly journal published by Cisco Systems for engineering professionals involved in designing, developing, and operating public and private internets and intranets. You can access the Internet Protocol Journal at this URL:

  http://www.cisco.com/en/US/about/ac123/ac147/about_cisco_the_internet_protocol_journal.html

- Training—Cisco offers world-class networking training. Current offerings in network training are listed at this URL:

  http://www.cisco.com/en/US/learning/index.html

Using the SESM Web Services Gateway

This chapter describes the Cisco Subscriber Edge Services Manager (SESM) Web Services Gateway and contains instructions on how to install, start, and use a Web Services Gateway (WSG) application. This chapter contains the following sections:

Introduction to the Web Services Gateway

The WSG is a SESM application that enables third-party web portal applications and subscriber management systems to integrate with the SESM and Service Selection Gateway (SSG) solution via web services.

Web Services

Web services are services that are available via the web. In a typical web services scenario, a business application sends a request to a service at a given URL. The service receives the request, processes it, and returns a response.

For more information on Java 2 Enterprise Edition (J2EE) web services used in SESM, see the java website.

In the context of SESM, the web services are a collection of services implemented by the WSG.

Any client application can interface with the WSGs using the Simple Object Access Protocol (SOAP) over HTTP. Web services are described using the Web Services Description Language (WSDL).

SOAP and WSDL

SOAP is a lightweight, XML-based remote procedure call (RPC) messaging protocol. It encapsulates the RPC data in Web service request and response messages typically transported using HTTP.

WSDL is an XML used to describe the interface methods and data contained in a Web service. In general, this information is contained in the file wsg/wsdl/*gateway_name*.wsdl. SOAP is the transport layer for the Microsoft .NET system and .NET technologies can be used to generate clients from the WSDL.

For a full description of the WSG WSDL file, see Appendix B, “Web Services Gateway Interface”.

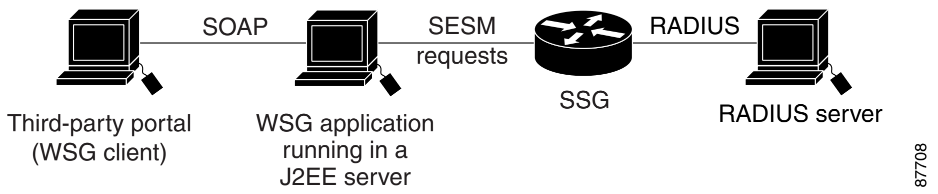
Software Binding

To get a specific language binding for a particular interface described in a WSDL file, you can use a WSDL compiler to compile the interface description file. This creates server and client stubs in the desired target language. These stubs are implemented by the WSGs to create a SOAP/HTTP server. SESM provides the Axis SOAP tools, located in the /redist directory.

Web Services Gateway

The WSG manages the interfaces between a client application and the SSG, or components of the solution. Figure 1-1 illustrates a typical WSG deployment topology.

Figure 1-1 WSG Deployment Topology


SOAP
SESM requests
RADIUS
Third-party portal (WSG client)
WSG application running in a J2EE server
SSG
RADIUS server
87708

The WSG is used for session control and service activation. It enables the WSG client interface to control and access the WSG for the following services:

- Authenticating, starting, and ending sessions on the WSG.
- Obtaining session status.
- Activating and deactivating services.

The WSG is described in detail in Chapter 2, "Web Services Gateway".

Configuring a WSG

This section describes the configuration tasks that you can perform with a SESM WSG.

The WSG is configured to listen on port 8100.

Note Do not open a new connection for every request. Use a connection pool for better performance.

Configuring the Jetty Container

To change the Jetty container configuration for a WSG application, edit the following file:

```
jetty
    config
        wsg.jetty.xml
```

Configuring a WSG Application

To modify the configuration of a WSG application, you can either:

- Access the MBeans through the Application Manager (AM) , or through Agent View. AM is configured to listen on port 8082. Agent View is configured to listen on port 8200 (*Web_Services_Gateway port number* + 100).

  or

- Manually edit the wsg.xml and dessauth.xml MBean configuration files:

  ```
  wsg
      config
         wsg.xml
         dessauth.xml
  ```

  The wsg.xml file configures the same MBeans as the SESM web portals.

  The dessauth.xml file configures the MBeans for the SPE component.

  For an explanation of the SESM MBeans, see *Cisco Subscriber Edge Services Manager Administration and Configuration Guide*.

  For more information about the AM, see *Cisco Subscriber Edge Services Manager Administration and Configuration Guide*.

Configuring WSG Security

SESM WSG supports Secure Sockets Layer (SSL) connection security. You can use SSL to verify the identity of the WSG service and encrypt all communication between it and the server. SSL is configured in the WSG by default.

To use SSL on the Server

- An SSL listener must be configured with a certificate for the server.

The WSG demo has a self-signed certificate for localhost, stored in the wsgkeystore file. The wsgkeystore file is used by the default configuration in wsg.jetty.xml.

To use SSL on the Client:

- When the client connects to the server, it receives a public certificate and a chain of trust must be established.

In a full deployment this is achieved by using a server certificate signed by a well known certificate authority whose certificate is in the trusted keystore provided by the Java Virtual Machine (JVM).

For the WSG demo installation, the wsgClient.sh script is configured to use the wsgkeystore file as the trusted keystore for the client. This keystore contains the self-signed certificate from the server, so the client will trust the certificate from that server.

SSL is used by the client when a https URL is passed as the endpoint. For example:

```
bin/wsgClient.sh https://localhost:8463/services/SESM
```

The passwords for the wsgkeystore are:

- store password: ciscostore.
- key password: ciscokey.

For information on how to generate self signed SSL certificates for testing environments see Appendix E in *Cisco Subscriber Edge Services Manager Administration and Configuration Guide*

Configuring WSG Authentication

The SESM WSG supports basic authentication of WSG clients. Basic authentication can be used to authenticate a WSG client to a particular service. Basic Authentication is configured in the WSG by default.

To enable basic authentication:

- The web server container must be configured with user credentials.
- The web application must be configured to require authentication for particular URLs.

Configuring the Web Server Container

1. The web server container must be configured with a realm of user credentials.

   A realm is a namespace in which user names and credentials are stored. The realm is configured to instruct the server how to authenticate user names and credentials. The realm can simply delegate the authentication to Lightweight Directory Access Protocol (LDAP) or a database. For example, you can have an LDAPRealm or a JDBCRealm.

   The Jetty server supports many types of realms. which can be configured in the web-jetty.xml file.
2. The web-jetty.xml file creates a simple HashUserRealm to configure usernames and properties.

   The HashUserRealm is a particular implementation of a realm for the Jetty server. It reads usernames and credentials from the wsgRealm.properties file and stores them in a hashmap.
3. The web-jetty.xml file is deployed in the *install_dir*/wsg/webapp/WEB-INF directory of a web application.

   The web-jetty.xml file is called to configure the Jetty-specific context of a particular web application file. By default, this file defines a single user name ‘wsg’ with password ‘wsg’ in the role ‘wsclient’. The role links to the security constraint configured in the web application for the /services/* and *.jws URL patterns.

Configuring the Web Application

You must configure the web application to require authentication for particular URLs.
In the web.xml file, the WSG web application has a security constraint defined for the directory /services/* and the java services *.jws. Only authenticated users with an assigned role of ‘wsclient’ are able to access these areas.

Note Basic authentication sends unencrypted passwords and should therefore be used only with SSL.

After authentication, WSG does not return a list of radius attributes, but it returns the text from the sa_sesm.wsdl file, which describes the interface of the WSG. This file is located in the wsg/wsdl directory of the SESM distribution. See Appendix B, “Web Services Gateway Interface” for the contents of the sa_sesm.wsdl file.

Disabling WSG Authentication

To disable authentication in a WSG, comment out the following section in *install_dir*/wsg/webapp/WEB-INF/web.xml:

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Any User</web-resource-name>
    <url-pattern>/services/*</url-pattern>
    <url-pattern>*.jws</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>wsclient</role-name>
  </auth-constraint>
</security-constraint>
```

Once authentication is disabled, any username and password are accepted when a connection request is made.

Overriding Buffer Settings

The WSG does not respond to authentication responses from the SSG if the receive buffer on the RADIUS listener is of limited size. To override the buffer settings in an application, edit the config file for that application found in <install directory>/<application name>/config/<application name>.xml. For example, for the WSG application, the config file is <install directory>/wsg/config/wsg.xml.

Find the following line in the config file (inside the SSG MBean):

```
<Call name="setGlobalAttribute"><Arg>THROTTLE</Arg><Arg>20</Arg></Call>
```

and place the following two lines below it:

```
<Call name="setGlobalAttribute"><Arg>RX_BUF_SIZE</Arg><Arg>4096</Arg></Call>
<Call name="setGlobalAttribute"><Arg>TX_BUF_SIZE</Arg><Arg>4096</Arg></Call>
```

You can increase the buffer size (4096) in these lines for the receive and transmit buffers. Recommended size is 32767.

The default buffer size after installing this patch is 4096. If this value is overridden with the value 0, the operating system default size is used.

On Linux systems the default buffer size is available in the following files:

* /proc/sys/net/core/rmem_default - default receive window
* /proc/sys/net/core/rmem_max - maximum receive window
* /proc/sys/net/core/wmem_default - default send window
* /proc/sys/net/core/wmem_max - maximum send window

On Unix systems you can run the following commands to get the default buffer size:

* ndd /dev/udp udp_recv_hiwat - maximum receive buffer size
* ndd /dev/udp udp_xmit_hiwat - Maximum transmit buffer size

Please note that these commands can vary based on the operating system.

Starting and Stopping a WSG Application

The SESM installation installs and configures the WSG application to run in a Jetty container.

To start a WSG application, run the startup script:

```
jetty
    bin
        startWSG.sh/cmd
```

To stop a WSG application, run the stop script:

```
jetty
    bin
        stopWSG.sh/cmd
```

These scripts accept all the options and parameters used by other SESM web applications.

For more information, refer to *Cisco Subscriber Edge Services Manager Web Portal Guide*.

CHAPTER 2

Web Services Gateway

This chapter provides a description of the Web Services Gateway (WSG) and describes the operations the interface. This chapter contains the following sections:

- Web Services Gateway Operations, page 2-1
- Host Key Support, page 2-5

Web Services Gateway Operations

The WSG enables session and service control, session status queries, and authentication, therefore permitting the client application to access web services.

Table 2-1 describes the operations supported by the WSG.

Table 2-1 The Web Services Gateway Operations

Operation Name	Description
authenticate	The WSG sends a request to the Service Selection Gateway(SSG) to create a session for the identities contained in the request. The SSG authenticates the identities with the associated credentials.
getStatus	Returns the session status.
serviceActivate	Activates a service on the SSG for the session identified by the identities.
serviceDeactivate	Deactivates the service on the SSG for the session identified by the identities.
endsession	Ends the session on the SSG for the session identified by the identities.

For a full list of the sa_sesm.wsdl file, including input and output elements for each operation, see Appendix B, "Web Services Gateway Interface".

Web Services Gateway Interface

WSG types:

The WSG interface uses the following types defined in the WSDL:

String - hostkey

This identifier is passed on to all WSG operations to identify the subscriber edge session being manipulated. The hostkey is an IP address and optional port number plus an optional list of attribute names and values in the format:

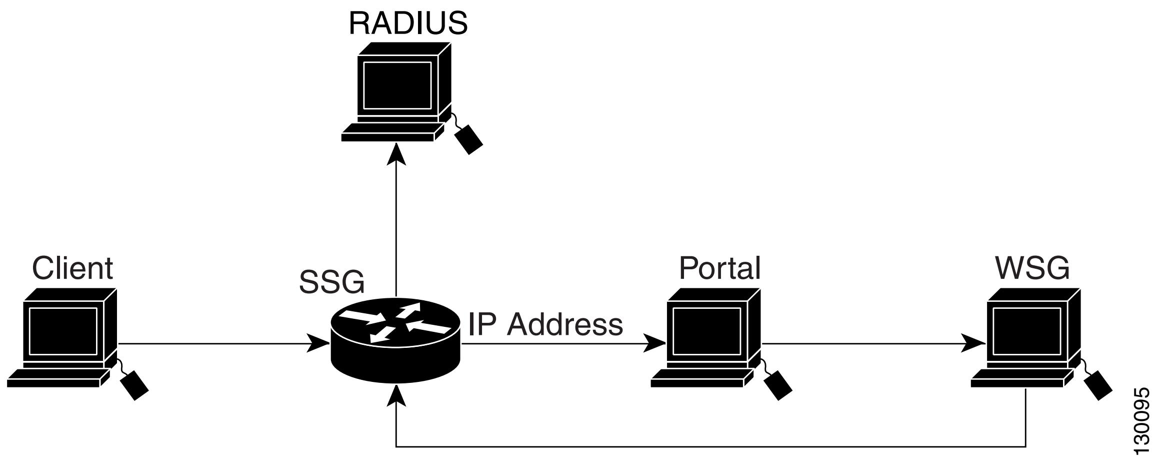
```
<ip> [ /<port> ] [ ; <name>=<value> ]*
```

The following are all legal hostkeys:

```
10.0.0.1
10.0.0.1;name1=val1
10.0.0.1;name1=val1;name2=val2;
10.0.0.1/4325
10.0.0.1/4325;name1=val1
10.0.0.1/4325;name1=val1;name2=val2;
```

The IP address is that of the subscriber connection to the third party portal as translated by the SSG:

Figure 2-1 IP address host key



The IP address may be the same as the client IP address or it may be translated, depending on the configuration of the SSG. If the SSG and SESM has been configured to use the PBHK feature, then the hostkey sent to the WSG must include the source port of the connection to the third party portal as translated by the SSG, separated from the ip with '/'. For example, 10.5.7.1/5678. The client application will need to obtain the source port from the client request.

Note This is the full port number and not the port-bundle number. Elsewhere in SESM you may see a client IP and port bundle number written as `<ip>:<portbundle>`. The port bundle is the actual port module of the bundle size. The WSG uses the port number so that the portal does not need to be configured with the bundle size used by the SSG.

The WSG uses the hostkey to derive a SESM session key and thus a SESM session. The following pseudo code written against the SESM API illustrates how the hostkey is translated:

```
HashMap keyAttrs = new HashMap();
keyAttrs.put(SESMSessionKey.RemoteAddress,remoteIPFromWSGHostKey);
keyAttrs.put(SESMSessionKey.RemotePort,remotePortFromWSGHostKey);

for (each attr value pair in WSG host key)
keyAttrs.put(attrNameFromHostKey,attrValueFromHostKey);

SESMSessionKey key =
AuthenticationAPI.getInstance().createSessionKey(keyAttrs);
```

See SESM documentation for details on meaningful attributes within a SESM Session Key (and thus within a WSG hostKey). These attributes are typically used to tunnel information about the server, proxy or other application specific information.

Complex type - Identity

An identity type is used in the WSG interface to communicate an identify of an authenticated user for a session or a service.

A WSG identity is a sequence of two strings:

- Type - The type of identity. For example; user, msisdn.
- Principal - The identity as a string. For example, a username or msisdn number.

When the server sends an Identity to the client, the map returned by com.cisco.sesm.core.model.Account.getIdentities() is converted to an array of WSG identities by converting each entry with the map key used as the Identity type and the map value is coerced to a com.cisco.sesm.types.Identity and getPrincipal() called to get the WSG principal.

Complex type - Credential

A Credential type is used by the WSG interface to send an Identity and credential from the client to the server. Typically this is a username and password. A WSG credential is a sequence of two fields:

- Identity - a WSG Identity
- Credential - a string holding the clear text credential (password)

When the WSG receives a credential from a WSG client, it is converted into a com.cisco.sesm.types.Identity instance. If the identity has a type of user, then a com.cisco.sesm.types.AccountIdentity is used.

Complex type - Service Status

The instantaneous state of a service is sent via the WSG interface as a Service Status type. The information includes activation state, service identities and some statistics:

- boolean—active.
- array of Identity—serviceIdentities.
- long—timeconnected.
- long—packetsToUser.
- long—packetsToService.
- long—bytesToUser.
- long—bytesToService.

Complex type - Status

The WSG interface uses this type to describe the status of a session:

- Array of Identity.
- Array of ServiceStatus.

Services groups are not represented in the WSG interface and all groups are expanded so that a flat list of real services results.

WSG Operations

`Status getStatus(hostkey)` - Get the current status of the session identified by the hostkey. If there is no SESM session for the hostkey, then a SESMSession is created. If a session cannot be created, null is returned.

`boolean authenticate(hostkey, Credential[])` - Authenticate a SESM session identified by the hostkey with the credentials.If the sesssion is already authenticated SESMSession.reauthenticate(i) is called. Otherwise SESMSession.authenticate(...) is called. If the session does not exist, it is first created.

`void endSession(hostkey)` - End the SESM session by calling SESMSession.deAuthenticate.

`boolean serviceActivate(hostkey,serviceName,Credential[] credentials)` - Activate a service on the SESM session. The session must be authenticated and the service is activated by a call to SESMSession.connectService. If service activation secedes, then true is returned, else false.

`void serviceDeactivate(hostkey,serviceName)` - Deactivates a service for the identified edge session by calling SESMSession.disconnectService(...)

Exception Handling

There is no explicit exception handling in the WSG interface. If a ModelException (normal operation exception) is thrown by SESM, it is logged by the WSG and a normal response is sent to the WSG client - if possible a negative result is indicated. If an exceptional exception occurs, then it is transmitted to the WSG client as a failure message. This indicates a fault in the operation of the system.

Host Key Support

The WSG supports the following host keys:

- IP address host key.
- Port-Bundle Host Key (PBHK).

Port-Bundle Host Key

The SSG PBHK feature enhances communication and functionality between the SSG and SESM by introducing a mechanism that uses the host source IP address and source port to identify and monitor subscribers.

To enable PBHK in the WSG client, the host object must be port mapped by the SSG. That is, the host object must be created on the SSG before the WSG client tries a connection.

For a detailed description of the PBHK feature, refer to the document *Cisco SSG Port-Bundle Host Key.*

Web Services Gateway Client

This appendix describes how to use the Web Services Gateway Client and the client interface that provides command-line access to the Web Services Gateway (WSG). This appendix contains the following sections:

- Using Web Services Clients, page A-1
- Running the Client Interface for the Web Services Gateway, page A-9

Using Web Services Clients

SESM provides a WSG client application which is useful for testing the WSG applications.

Any client application can interface with the WSGs using the Simple Object Access Protocol (SOAP) over HTTP. SOAP allows disparate technologies to be used to implement client and server technologies. The SESM WSG provides a service that has been implemented in Java. However web services clients may be implemented in any programming language that supports SOAP over the HTTP standard.

This is achieved by using the decoupled SOAP protocol for the interface to the WSG. This SESM WSG interface is described in the Web Services Description Language (WSDL). This description can be used to generate client code in a programming language of the users' choice.

The client code generated from the WSDL translates the SESM WSG calls from the client's native programming language into standardized SOAP messages and sends them to the WSG over the HTTP protocol. The client code also receives the responses from the WSG and translates the standard SOAP messages back to the client's programming language.

The SESM WSG uses the apache Axis 1.1 implementation of SOAP, which supports bindings to the Java, C and C++ languages. (For details on Axis implementation of SOAP see the Apache web site) You may use other implementations of SOAP to generate the client code if other programming languages or environments are required (e.g..NET).

Note You may face interoperability problems when using client implementations other than Axis 1.1 as SOAP is an emerging standard. If such problems are encountered please contact your Cisco support representative for the latest updates.

Implementing a web services client using WSDL involves the following tasks:

- Generate client stubs from WSDL.
- Compile client stubs.
- Write client application using the client stubs.
- Link and execute client application + client stubs + axis runtime libraries.

Note Some environments, for example .Net, may automate all or part of this process.

Client Code

Typically two types of code need to be generated from the WSDL for use by the client application programmer. These are the interface types and stubs. An interface type is a data structure that is sent and received over the SOAP protocol bound to the clients programming language. For example, the WSG WSDL defines the type Identity in the WSDL as follows:

```
<complexType name="Identity"
          <sequence
            <!-- The type of the identity. --
          <!-- Known values are: "IP Address", "MAC Address",
  "VPI/VCI" --
            <!-- "SSG sub interface", "user name" and "msisdn" --
            <element name="type" nillable="false"
  type="soapenc:string"/
            <!-- The value of the identity. --
            <element name="principal" nillable="false"
  type="soapenc:string"/
          </sequence
        </complexType
```

If C is the client programming language, then a C strut is generated to hold the data of an Identity and to allow it to be passed to and from the interface:

```
typedef struct IdentityTag {
          xsd__string type;
          xsd__string principal;
      } Identity;
```

If C++ is the client programming language, the a class is generated instead of the structure:

```
class Identity
     {
       public:
          xsd__string type;
          xsd__string principal;
          Identity();
          ~Identity();
     };
```

The code associated with this class declaration is also generated. Likewise if the client programming language is Java, then a java class is generated to hold the same data:

```
public class Identity  implements java.io.Serializable {
      private java.lang.String type;
      private java.lang.String principal;

      public Identity() {}
      public java.lang.String getType() { return type; }
      public void setType(java.lang.String type) { this.type = type; }
      public java.lang.String getPrincipal() { return principal; }
      public void setPrincipal(java.lang.String principal) {
  this.principal = principal; }

      // ... plus static utitity methods
    }
```

In addition to the interface types, the interface stubs are also generated. A stub is a programming interface that appears like the interface defined in the WSDL, but instead of providing an implementation of the actual behavior, the stub implementation uses SOAP to communicate with the real implementation on the WSG server. In both C++ and Java, the generated stub is an interface with a signature derived from the WSDL and a class that implements that interface.

The generated interface in C++ is:

```
class SESM
    {
      private:
          Call* m_pCall;
      public:
          SESM();
          virtual ~SESM();
          void authenticate(xsd__string Value0,Credential_Array
  Value1,xsd__boolean Value2);
          void endSession(xsd__string Value0);
          void getStatus(xsd__string Value0,Status* Value1);
          void serviceActivate(xsd__string Value0,xsd__string
  Value1,Credential_Array Value2,xsd__boolean Value3);
          void serviceDeactivate(xsd__string Value0,xsd__string Value1);
    };
```

The generated interface in Java is:

```
public interface SESM extends Remote
    {
      public boolean authenticate(String hostkey, Credential[]
  credentials) throws RemoteException;
      public void endSession(String hostkey) throws RemoteException;
      public Status getStatus(java.lang.String hostkey) throws
  RemoteException;
      public boolean serviceActivate(String hostkey, String
  service, Credential[] credentials) throws RemoteException;
      public void serviceDeactivate(String hostkey, String
  service) throws RemoteException;
    }
```

If the client language is C, then functions are defined and implemented for each interface method:

```
extern void authenticate(void* pStub, xsd__string Value0,
  Credential_Array Value1, xsd__boolean Value2);
    extern void endSession(void* pStub, xsd__string Value0);
    extern void getStatus(void* pStub, xsd__string Value0,
  Status* Value1);
    extern void serviceActivate(void* pStub, xsd__string
  Value0, xsd__string Value1, Credential_Array Value2,
  xsd__boolean Value3);
    extern void serviceDeactivate(void* pStub, xsd__string
  Value0, xsd__string Value1);
```

Generating the Client Code

To generate interface types and stubs, implementation of SOAP provides utility programs that read the WSDL and generate the required files for both the server side and the client side.

For the Axis 1.1 SOAP implementation, this utility is a java class: org.apache.axis.wsdl.WSDL2Java that generates java stubs and org.apache.axis.wsdl.wsdl2ws.WSDL2Ws. To run these utilities, the java classpath must be set to include the jar files indicated by Axis documentation.

Note Currently, versions later than Axis 1.1 do not generate C++ code correctly.

The types and stubs may then be generated with the following commands:

Client Type	Command
C clients	java org.apache.axis.wsdl.wsdl2ws.WSDL2Ws sesm.wsdl -lc -sclient
C++ clients	java org.apache.axis.wsdl.wsdl2ws.WSDL2Ws sesm.wsdl -lc++ -sclient
Java clients	java org.apache.axis.wsdl.WSDL2Java sesm.wsdl

See Appendix B, “Web Services Gateway Interface” for information about the sesm.wsdl file, which provides a description of the WSG interface distributed with SESM.

Note In order to generate code in C or C++, you must uncomment the service definition section at the end of the sesm.wsdl file.

Using the Generated Client Code

Each SOAP implementation is different in how its generated stubs are to be used by application developers. Refer appropriate documentation for details. However, all implementations follow the same basic steps:

- Initialize the SOAP runtime and configure the SOAP transport.
- Create the stub instance.
- Bind the stub instance to a URL defining the location of the service.
- Set any additional transport authenticate/security properties.
- Use the stub instance.

Using java as an example language, the following code creates a SESM stub and calls the authenticate method.

```
// Initialize the SOAP runtime and configure the SOAP transport
        EngineConfiguration defaultConfig =
  EngineConfigurationFactoryFinder.newFactory().getClientEngineConfig();
        SimpleProvider config = new SimpleProvider(defaultConfig);
        SimpleTargetedChain c = new SimpleTargetedChain(new
  org.apache.axis.transport.http.HTTPSender());
        config.deployTransport("http", c);
        Service service = new Service(config);

        // create the stub instance and bind to URL
        String endpointURL="http://wsghost:8080/services/SESM";
        SESM sesm = new SesmSoapBindingStub(endpointURL,service);

        // Configure authentication
        ((SesmSoapBindingStub)sesm).setUsername(service_username);
        ((SesmSoapBindingStub)sesm).setPassword(service_password);

        // use the stub instance
        Identity id = new Identity();
        id.setType(IdentityType.USER.toString());
        id.setPrincipal(user_username);
        Credential[] creds = {new Credential()};
        creds[0].setIdentity(id);
        creds[0].setCredential(user_password);
        sesm.authenticate(user_hostkey,creds));
```

Using the Axis WSG Stubs

The SESM solution contains java stubs of the WSG interface that have been generated from the SESM WSDL. These stubs may be used by a client program written in Java to access the WSG. The stubs are generated by Axis 1.1. Axis documentation should be consulted for the full reference on using these stubs.

Required Classes

The client code needs to import the following classes generated by Axis:

```
import com.cisco.sesm.wsg.common.Credential;
import com.cisco.sesm.wsg.common.Identity;
import com.cisco.sesm.wsg.common.ServiceStatus;
import com.cisco.sesm.wsg.common.SesmSoapBindingStub;
import com.cisco.sesm.wsg.common.Status;
```

These classes are contained within the com.cisco.sesm.wsg.jar file which must be on the CLASSPATH of the java compiler and the java runtime for the client.

The following Axis classes are also needed:

```
import org.apache.axis.EngineConfiguration;
import org.apache.axis.SimpleTargetedChain;
import org.apache.axis.client.Service;
import org.apache.axis.configuration.EngineConfigurationFactoryFinder;
import org.apache.axis.configuration.SimpleProvider;
import org.apache.axis.Handler;
```

These classes are contained in the axis.jar file from apache, which also must be available on the CLASSPATH of the java compiler and the java runtime for the client.

To run the client, the following jars (bundled with Axis) must also be on the CLASSPATH:

- commons-discovery.jar.
- commons-logging.jar.
- jaxrpc.jar.
- mail.jar.
- saaj.jar.
- wsdl4j.jar.

Axis Initialization

In order to use an Axis generated stub, the Axis runtime service and transport must be initialized. For full details of this, see the Axis documentation.

The following code initializes the Axis service:

```
// Setup configuration for AxisClient
   EngineConfiguration defaultConfig =

  EngineConfigurationFactoryFinder.newFactory().getClientEngineConfig();
   SimpleProvider config = new SimpleProvider(defaultConfig);
   SimpleTargetedChain c =
     new SimpleTargetedChain(new
  org.apache.axis.transport.http.HTTPSender());
   config.deployTransport("http", c);
   Service service = new Service(config);
```

This configuration uses the default Axis HTTP transport, which is a simple HTTP client using non-persistent connections. The com.cisco.sesm.wsg.jar bundles an alternative HTTP transport that uses the JVM’s URLConnection class, which may use persistent connections and other more efficient techniques.

This alternative transport can be used by replacing the org.apache.axis.transport.http.HTTPSender class in the code above with org.apache.axis.transport.http.HttpURLConnectionSender. Alternative transports are also available from Axis and other sources that use the Jakarta HttpClient transport.

Stub Initialization

Once the axis service is initialized, an instance of the SESM WSG stub can be created and initialized with the following code:

```
sesm = new SesmSoapBindingStub(endpointURL,service);
sesm.setUsername(username);
sesm.setPassword(password);
```

The endpointURL is a string URL that gives the location of the SESM WSG server. Examples of non-SSL and SSL endpoint URLs are:

http://sesmhost:8100/services/SESM

https://sesmhost:8463/services/SESM

The username and password set here are not those of the actual SESM users. Rather they are a username and password that has been allocated to the WSG client itself so that it can authenticate with the SESM WSG service. See Configuring WSG Authentication, page 1-4 for information on configuring these usernames and passwords.

Typically a single instance of the SESM WSG stub class will be used by a WSG client and requests for multiple users will be sent over the shared instance.

For optimum throughput, the number of user requests that share a single sesm stub instance should be limited. This limit needs to be determined by benchmarking the actual hardware and server used with realistic requests.

Using the WSG Stub

Once initialized, the WSG stub instance may be used to make calls on the WSG server as it is a local class to the client. To make some WSG calls, the generated data type classes must be created and/or decoded. For example the following code creates an authentication request for a given hostkey, username and password:

```
public void authenticate(SESM sesm, String hostkey, String
 username, String password)
    throws RemoteException
  {
      Identity id = new Identity();
      id.setType("user");
      id.setPrincipal(username);

      Credential[] credential = {new Credential()};
      credential[0].setIdentity(id);
     credential[0].setCredential(password);

      sesm.authenticate(hostkey,credential);
   }
```

This code will either succeed in authenticating the user for the given hostkey, or it will throw a RemoteException, which will wrap the real exception that should explain the cause of the failure. The wrapped exception may be retrieved with the getCause() method of RemoteException.

The following code illustrates how the getStatus method can be called and the resulting information displayed on stdout:

```
      public void getStatus(SESM sesm, String hostkey)
          throws RemoteException
      {
          System.out.println("Status of "+hostkey+":");

          // get the status of the host key;
          Status status = sesm.getStatus(hostkey);

          if (status != null)
          {
            // report on all identities for hostkey
            Identity[] ids = status.getIdentities();
            for(int i=0; ids!=null && i<ids.length; i++)
              System.out.println(" Identity
  "+ids[i].getType()+": "+ids[i].getPrincipal());

            // report on all services for hostkey
            ServiceStatus[] statii = status.getServices();
            for(int i=0; statii!=null && i<statii.length; i++)
            {
              System.out.print("Service "+statii[i].getService());
              if (statii[i].isActive())
              {
                 System.out.print(": ON ");
                 ids = statii[i].getServiceIdentities();
                 char s='(';
                 for(int j=0; ids!=null && j<ids.length; j++)
                 {

  System.out.print(s+ids[j].getType()+":"+ids[j].getPrincipal());
                    s=',';
                 }

                 System.out.print(")
  time="+statii[i].getTimeConnected());
                 System.out.print(" b2u="+statii[i].getBytesToUser());
                 System.out.print("
  b2s="+statii[i].getBytesToService());
                 System.out.print(" p2u="+statii[i].getPacketsToUser());
                 System.out.println("
  p2s="+statii[i].getPacketsToService());
               }
               else
                 System.out.println(": off");
             }
           }
         }
      }
```

Running the Client Interface for the Web Services Gateway

The WSG client application is useful for testing the WSG applications. You can activate commands with the client that will communicate with WSG. The client interface provides a command-line interface for interacting with the WSG using SOAP remote procedure calls (RPC).

This interface is intended to demonstrate and test the functionality of the WSG. In order to deploy the WSG, you must develop your own SOAP client interface to replace the demonstration client interface.

The demonstration client interface script is located in here:

```
wsg
    bin
        wsgClient
```

Starting the WSG Client

To start the WSG client, perform the following procedure:

Step 1 Enter:

`wsgClient [`*`endpoint`*`]`

Where *endpoint* is either

`http://`*`WSGhost`*`:8100/services/SESM`

To connect using a non-secure port.

or

`https://`*`WSGhost`*`:8463/services/SESM`

To connect using a secure port.

Step 2 At the prompt enter a user name and password for basic authentication between the WSG client and the WSG server.

`WSG Username:`
The default user name is `wsg`.

`WSG Password:`
The default password is `wsg`.

Note To disable authentication from the WSG client startup, see Disabling WSG Authentication, page 1-5.

Once you have been authenticated, the WSG client command-line prompt is displayed:

```
wsg>
```

At the prompt, enter **help** to display available commands. At subsequent prompts, enter any of the commands listed in Table A-1.

Table A-1 The Web Services Gateway Client Commands

Command	Description
auth	Authenticates the user request, and then creates a user's host object on the SSG from data stored in the Security Policy Engine (SPE) or RADIUS. This information contains the user's shape, including username, host-id, and the services to which the user is subscribed.
get	Gets a host object from SPE after user authentication.
act	Activates or reactivates a service in a user's host-id. For example, the command **`act iptv - -`** activates iptv service with no username or password.
dea	Deactivates a service from a user's host-id. For example, the command **`dea iptv`** deactivates iptv service.
end	Ends a session by removing the host object from the SSG.
host	Sets the host key.
h	Displays help information.
fork *n*	Run the remainder of the command line *n* times in parallel.
loop *n*	Run the remainder of the command line *n* times in series.
close	Close the command line input so that client exits at the end of test.
query	Query the state of the host key session.

WSG Client Command Examples

This section provides examples of the WSG commands and their output.

Setting the Host

Use the **host** or **hostkey** command to set the IP address and port number of the WSG client.

Setting the Hostkey for an IP Address Host Key Session

Step 1 Use the **hostkey** command to set the host for an IP address host key session:

```
wsg> hostkey subscriber_IP_address
```

The SSG must be able to route this IP address through a configured downlink interface when it creates the edge session for the subscriber.

For example:

```
wsg> hostkey 209.165.200.225
hostkey=209.165.200.225
```

Step 2 Use the ***`hostkey %`*** command to set a random host key.

Setting the Host for a Port-Bundle Host Key Session

Step 1 Obtain hostkey information from the SSG, as follows:

To get the IP address map of the subscriber from the SSG, use the following command:

```
ssg>show ssg host
1: 64.103.64.210  [Host-Key 1.1.1.5:65]
```

In this example, the IP address map of the host key is 1.1.1.5 and the port bundle is 65.

Step 2 Get the port number that the SSG has mapped to the subscriber, using the `ssg port-map` command with the parameters that were returned by the `show ssg host` command.

For example:

```
ssg>sh ssg port-map ip 1.1.1.5 port 65

State = IN-USE
Subscriber Address = 64.103.64.210
Downlink Interface = FastEthernet1/0

Port-mappings: -

Subscriber Port:    2257                Mapped Port:    1040
```

In this example, the mapped port for the subscriber is 1040.

Step 3 Set the host for a Port-Bundle Host Key(PBHK) session on the WSG session as follows once you have the IP address map and the port map information you can:

```
wsg> host IP_address_map/port_map
```

For example:

```
wsg> host 1.1.1.5/1040
hostkey = 1.1.1.5/1040
```

Authenticating a User

Use the `auth` command to authenticate the subscriber with the SSG.

```
wsg> auth username password
```

For example:

```
wsg> auth golduser cisco
authenticate=true
```

Use the `auth % cisco` command to authenticate a random user 0 to user 45.

Activating and Deactivating a Service

Use the **get** command to display the status of the user account and the services the user is currently subscribed to. The following parameters are provided for each active service:

- time—time that the service has been active for the session (in seconds).
- b2u—number of bytes sent downstream for the session.

- b2s—number of bytes sent upstream for the session.
- p2u—number of packets sent downstream for the session.
- p2s—number of packets sent upstream for the session.

For example:

```
wsg> get
Identity user: golduser
Service games: off
Service dynamic_bandwidth_selection: off
Service Internet-Basic: off
Service discount_shopping: off
Service Internet-Premium: ON (user:) time=625 b2u=5238 b2s=1020 p2u=56 p2s=22

Service corporate: off
Service Internet-Standard: off
```

Use the **dea** command to deactivate a service. For example:

```
wsg> dea Internet-Premium
deactivate Internet-Premium
```

Use the **get** command to verify that the service was deactivated. For example:

```
wsg> get
Identity user: golduser
.
.
.
Service Internet-Premium: off
```

Use the **act** command to activate the service. For example:

```
wsg> act Internet-Premium
activate Internet-Premium = true
```

Use the `get` command to verify that the service was reactivated. For example:

```
wsg> get
Identity user: golduser
.
.
.
Service Internet-Premium: ON (user:) time=625 b2u=5238 b2s=1020 p2u=56 p2s=22
```

Ending a WSG Client Session

Use the **end** command to end the current session and use the **quit** command to exit the shell.

```
wsg> end
endSession
wsg> quit
user1>
```

APPENDIX B

Web Services Gateway Interface

This appendix contains the full text of the sesm.wsdl file, which provides a description of the Web Services Gateway (WSG) interface.

The sesm.wsdl file describes the interface of the WSG, in the Web Service Description Language (WSDL) format. This describes the contents of the sesm.wsdl file, located in the wsg/wsdl directory of the SESM distribution.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Copyright (c) 2002, 2003 by Cisco Systems, Inc. All rights reserved. -->

<wsdl:definitions
 targetNamespace="http://common.wsg.sesm.cisco.com"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:apachesoap="http://xml.apache.org/xml-soap"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:intf="http://common.wsg.sesm.cisco.com"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:types>
    <schema
     xmlns="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://common.wsg.sesm.cisco.com">
     <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>


      <!-- A SESM Identity Type -->
      <complexType name="Identity">
        <sequence>
          <!-- The type of the identity. -->
      <!-- Known values are: "IP Address", "MAC Address", "VPI/VCI" -->
          <!-- "SSG sub interface", "user name" and "msisdn" -->
          <element name="type" nillable="false" type="soapenc:string"/>
          <!-- The value of the identity. -->
          <element name="principal" nillable="false" type="soapenc:string"/>
        </sequence>
      </complexType>

      <!-- Collection of SESM Identities -->
      <complexType name="ArrayOfIdentity">
        <complexContent>
          <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType" wsdl:arrayType="intf:Identity[]"/>
          </restriction>
        </complexContent>
      </complexType>
      <element name="ArrayOfIdentity" nillable="true" type="intf:ArrayOfIdentity"/>

      <!-- A SESM Credential Type -->
      <complexType name="Credential">
        <sequence>
          <!-- The type of the identity. Known values are "user" and "msisdn" -->
          <element name="identity" nillable="false" type="intf:Identity"/>
          <!-- The credential of the identity. Normally a password or msisdn -->
          <element name="credential" nillable="false" type="soapenc:string"/>
        </sequence>
      </complexType>

      <!-- Collection of SESM Credentials -->
      <complexType name="ArrayOfCredential">
        <complexContent>
          <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType" wsdl:arrayType="intf:Credential[]"/>
          </restriction>
        </complexContent>
      </complexType>
      <element name="ArrayOfCredential" nillable="true" type="intf:ArrayOfCredential"/>


      <!-- Status type of a SESM Service -->
      <complexType name="ServiceStatus">
        <sequence>
          <element name="service" nillable="false" type="soapenc:string"/>
          <element name="active" nillable="false" type="xsd:boolean"/>
          <element name="serviceIdentities" nillable="true" type="intf:ArrayOfIdentity"/>
          <element name="timeConnected" nillable="false" type="xsd:long"/>
          <element name="packetsToUser" nillable="false" type="xsd:long"/>
          <element name="packetsToService" nillable="false" type="xsd:long"/>
          <element name="bytesToUser" nillable="false" type="xsd:long"/>
          <element name="bytesToService" nillable="false" type="xsd:long"/>
        </sequence>
      </complexType>

      <!-- Collection of service status types -->
      <complexType name="ArrayOfServiceStatus">
        <complexContent>
          <restriction base="soapenc:Array">
            <attribute ref="soapenc:arrayType" wsdl:arrayType="intf:ServiceStatus[]"/>
          </restriction>
        </complexContent>
      </complexType>
      <element name="ArrayOfServiceStatus" nillable="true"
type="intf:ArrayOfServiceStatus"/>


      <!-- Status type of a SESM Session -->
      <complexType name="Status">
        <sequence>
          <element name="identities" nillable="true" type="intf:ArrayOfIdentity"/>
          <element name="services" nillable="true" type="intf:ArrayOfServiceStatus"/>
        </sequence>
      </complexType>

    </schema>
  </wsdl:types>

  <!-- A boolean response.
   * Utility response message used to indicate success or failure of a
   * request
```

```
 -->
<wsdl:message name="booleanResponse">
  <wsdl:part  name="return" type="xsd:boolean"/>
</wsdl:message>

<!--
 * SESM Authentication request.
 * This method is used if authentication is required for the edge session.
 * This is not needed if SSO is being used. Authentication is valid until
 * the edge session is ended.
 *   hostkey    : a String encoding the IP and optional port of
 *                the remote end of the connection. This is used to identify
 *                the edge session and the matching SESM session.
 *                The format if PBHK is being used is "remoteIP/remotePort",
 *                else just the remote IP is passed as "remoteIP".
 *   credentials : Array of credentials to use to authenticate the session.
 *                If no credentials are passed, this request acts as
 *                authentication status query.
 -->
<wsdl:message name="authenticateRequest">
  <wsdl:part  name="hostkey" type="soapenc:string"/>
  <wsdl:part  name="credentials" type="intf:ArrayOfCredential"/>
</wsdl:message>


<!--
 * End Edge and SESM session request.
 * End the Edge session and any associated SESM sessions.
 *   hostkey    : a String encoding the IP and optional port of
 *                the remote end of the connection. This is used to identify
 *                the edge session and the matching SESM session.
 *                The format if PBHK is being used is "remoteIP/remotePort",
 *                else just the remote IP is passed as "remoteIP".
 -->
<wsdl:message name="endSessionRequest">
  <wsdl:part  name="hostkey" type="soapenc:string"/>
</wsdl:message>


<!--
 * SESM service status request.
 * Returns the status of all subscribed services for the session.
 *   hostkey    : a String encoding the IP and optional port of
 *                the remote end of the connection. This is used to identify
 *                the edge session and the matching SESM session.
 *                The format if PBHK is being used is "remoteIP/remotePort",
 *                else just the remote IP is passed as "remoteIP".
 -->
<wsdl:message name="getStatusRequest">
  <wsdl:part  name="hostkey" type="soapenc:string"/>
</wsdl:message>

<!-- SESM Service Status response.
 * Account name and a Collection of SESM service status objects for the
 * requested edge session.
 -->
<wsdl:message name="getStatusResponse">
  <wsdl:part  name="status" type="intf:Status"/>
</wsdl:message>

<!--
 * Activate SESM service.
 * Activates a service for the identified edge session
 *   hostkey    : a String encoding the IP and optional port of
 *                the remote end of the connection. This is used to identify
 *                the edge session and the matching SESM session.
 *                The format if PBHK is being used is "remoteIP/remotePort",
 *                else just the remote IP is passed as "remoteIP".
 *   service    : The name of the service.
 *   credentials : Array of credentials to use to activate to service. May be
 *                null or empty for unauthenticated services.
 -->
<wsdl:message name="serviceActivateRequest">
  <wsdl:part  name="hostkey" type="soapenc:string"/>
  <wsdl:part  name="service" type="soapenc:string"/>
  <wsdl:part  name="credentials" type="intf:ArrayOfCredential"/>
</wsdl:message>

<!--
 * Deactivate  SESM service.
 * Deactivates a service for the identified edge session
 *   hostkey    : a String encoding the IP and optional port of
 *                the remote end of the connection. This is used to identify
 *                the edge session and the matching SESM session.
 *                The format if PBHK is being used is "remoteIP/remotePort",
 *                else just the remote IP is passed as "remoteIP".
 *   service    : The name of the service.
 -->
<wsdl:message name="serviceDeactivateRequest">
  <wsdl:part  name="hostkey" type="soapenc:string"/>
  <wsdl:part  name="service" type="soapenc:string"/>
</wsdl:message>




<wsdl:portType name="SESM">
  <wsdl:operation name="authenticate"
                 parameterOrder="hostkey credentials">
    <wsdl:input   name="authenticateRequest"
                 message="intf:authenticateRequest"/>
    <wsdl:output  name="booleanResponse"
                 message="intf:booleanResponse"/>
  </wsdl:operation>


  <wsdl:operation name="endSession"
                 parameterOrder="hostkey">
    <wsdl:input   name="endSessionRequest"
                 message="intf:endSessionRequest"/>
  </wsdl:operation>


  <wsdl:operation name="getStatus"
                 parameterOrder="hostkey">
    <wsdl:input   name="getStatusRequest"
                 message="intf:getStatusRequest"/>
    <wsdl:output  name="getStatusResponse"
                 message="intf:getStatusResponse"/>
  </wsdl:operation>


  <wsdl:operation name="serviceActivate"
                 parameterOrder="hostkey service credentials">
    <wsdl:input   name="serviceActivateRequest"
                 message="intf:serviceActivateRequest"/>
    <wsdl:output  name="booleanResponse"
                 message="intf:booleanResponse"/>
    </wsdl:operation>


    <wsdl:operation name="serviceDeactivate"
                    parameterOrder="hostkey service">
      <wsdl:input   name="serviceDeactivateRequest"
                    message="intf:serviceDeactivateRequest"/>
    </wsdl:operation>

  </wsdl:portType>


  <wsdl:binding name="sesmSoapBinding" type="intf:SESM">
      <wsdlsoap:binding style="rpc"
                        transport="http://schemas.xmlsoap.org/soap/http"/>

      <wsdl:operation name="authenticate">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="authenticateRequest">
            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://common.wsg.sesm.cisco.com"/>
         </wsdl:input>
         <wsdl:output name="booleanResponse">
            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://common.wsg.sesm.cisco.com"/>
         </wsdl:output>
      </wsdl:operation>


      <wsdl:operation name="endSession">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="endSessionRequest">
            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://common.wsg.sesm.cisco.com"/>
         </wsdl:input>
      </wsdl:operation>



      <wsdl:operation name="getStatus">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="getStatusRequest">
            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://common.wsg.sesm.cisco.com"/>
         </wsdl:input>
         <wsdl:output name="getStatusResponse">
            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://common.wsg.sesm.cisco.com"/>
         </wsdl:output>
      </wsdl:operation>



      <wsdl:operation name="serviceActivate">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="serviceActivateRequest">
            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://common.wsg.sesm.cisco.com"/>
```

```
         </wsdl:input>
         <wsdl:output name="booleanResponse">
            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://common.wsg.sesm.cisco.com"/>
         </wsdl:output>
      </wsdl:operation>

      <wsdl:operation name="serviceDeactivate">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="serviceDeactivateRequest">
            <wsdlsoap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://common.wsg.sesm.cisco.com"/>
         </wsdl:input>
      </wsdl:operation>


  </wsdl:binding>


<!--
   <wsdl:service name="SESMService">
      <wsdl:port name="SESM" binding="intf:sesmSoapBinding">
         <wsdlsoap:address location="http://localhost:8080/services/SESM"/>
      </wsdl:port>
   </wsdl:service>
-->

</wsdl:definitions>
```

Note In order to generate code in C or C++, you must uncomment the service definition section at the end of the file.

INDEX