Oracle® Communications Convergence

Security Guide

Release 3.0.1

E56611-01

May 2015

ORACLE®

Oracle Communications Convergence Security Guide, Release 3.0.1

E56611-01

Contents

Preface

This guide provides guidelines and recommendations for setting up Oracle Communications Convergence in a secure configuration.

Audience

This document is intended for installers, system administrators, or software technicians who work with Convergence. This guide assumes you are familiar with the following concepts:

- Oracle GlassFish Server administration
- Convergence Server administration
- System administration and networking
- General deployment architectures

Related Documents

For more information, see the following documents:

- *Convergence Installation and Configuration Guide*: Describes the requirements for installing Convergence.
- *Convergence System Administrator's Guide*: Describes how to monitor and manage Convergence.
- *Convergence Customization Guide*: Describes how to customize the look and feel of Convergence.
- *Convergence Release Notes*: Describes any known issues for Convergence.

Document Revision History

The following table lists the revision history for this guide.

Version	Date	Description
E56611-01	May 2015	3.0.1 GA release.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

1

Convergence Security Overview

This chapter provides an overview of security for Oracle Communications Convergence.

Basic Security Considerations

The following principles are fundamental to using any application securely:

1. **Keep software up to date**. This includes the latest product release and any patches that apply to it.
2. **Limit privileges as much as possible**. Users should be given only the access necessary to perform their work. User privileges should be reviewed periodically to determine relevance to current work requirements.
3. **Monitor system activity**. Establish who should access which system components, how often they should be accessed, and who should monitor those components.
4. **Install software securely**. For example, use firewalls, secure protocols (such as SSL), and secure passwords. See "Performing a Secure Convergence Installation" for more information.
5. **Learn about and use Convergence security features**. See "Implementing Convergence Security" for more information.
6. **Use secure development practices**. For example, take advantage of existing database security functionality instead of creating your own application security.
7. **Keep up to date on security information**. Oracle regularly issues security-related patch updates and security alerts. You must install all security patches as soon as possible. See the Oracle Critical Patch Updates and Security Alerts web site:

   `http://www.oracle.com/technetwork/topics/security/alerts-086861.html`

Understanding the Convergence Environment

When planning your Convergence environment, consider the following:

- Which resources require protection?

  For example:

  - Convergence
  - Protocols, such as HTTP, WMAP, WCAP, LDAP, XMPP, WABP, NABP

  - Dependent resources, such as Oracle GlassFish Server, Directory Server, Index Search Service, Messaging Server, Calendar Server, Instant Messaging Server, WebRTC Session Controller, Oracle Outside In Transformation Server

- From whom do the resources require protection?

  In general, resources must be protected from everyone on the Internet. But should the Convergence deployment be protected from employees on the intranet in your enterprise? Should your employees have access to all resources within the GlassFish Server environment? Should the system administrators have access to all resources? Should the system administrators be able to access all data? You might consider giving access to highly confidential data or strategic resources to only a few well trusted system administrators. On the other hand, perhaps it would be best to allow no system administrators access to the data or resources.

- What happens if protections on strategic resources fail?

  In some cases, a fault in your security scheme is easily detected and considered nothing more than an inconvenience. In other cases, a fault might cause great damage to companies or individual clients that use Convergence. Understanding the security ramifications of each resource help you protect it properly.

Overview of Convergence Server Security

Figure 1–1 shows the various components that comprise a Convergence deployment. Each installed or integrated component requires special steps and configurations to ensure complete system security.

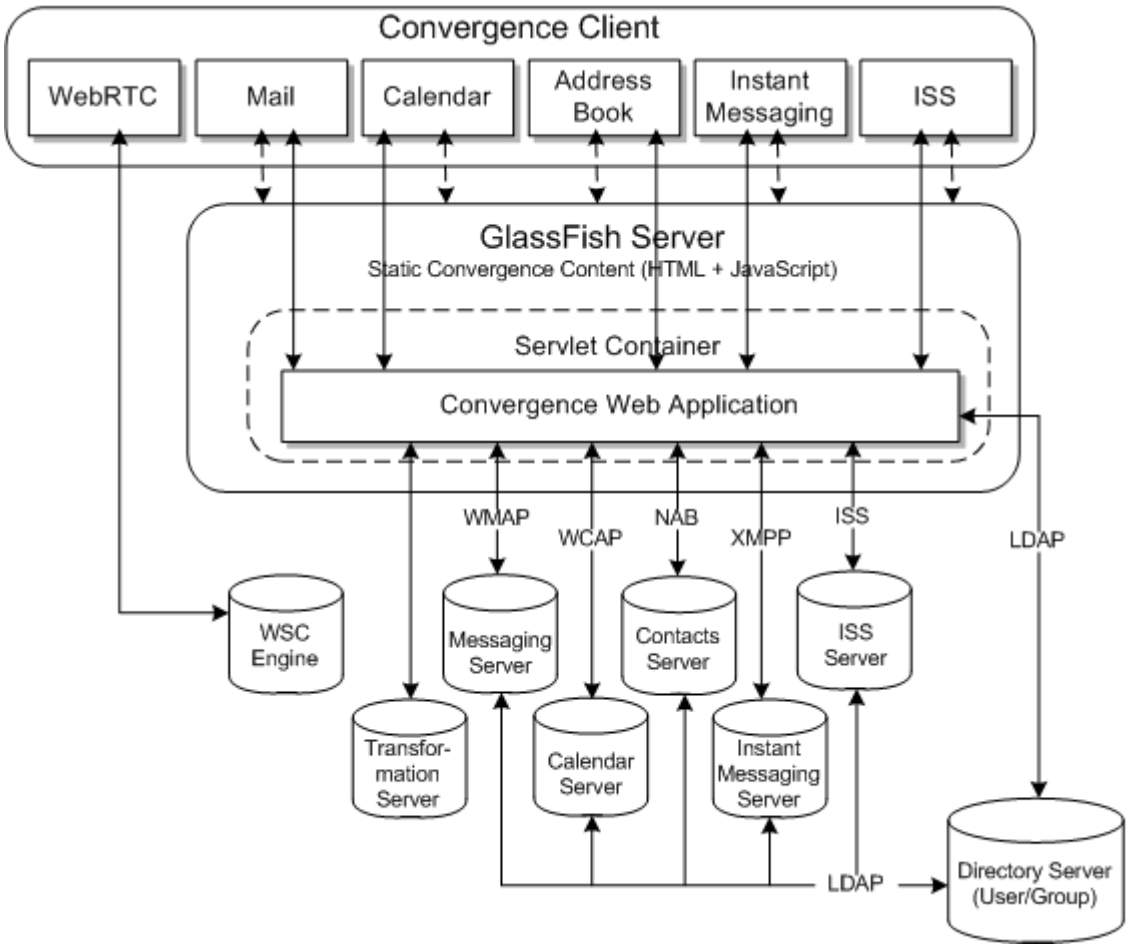
The top layer shows the services provided by Convergence. The middle layer represents the Convergence server itself, deployed to an Oracle GlassFish Server domain. The bottom layer shows the dependencies that the Convergence server has on other applications to provide its services and features.

Convergence consists of the following core services:

- Service Proxies
- XMPP over HTTP Gateway
- Address Book Service
- Authentication & Authorization
- SSO (Oracle Access Manager/Messaging SSO)
- Configuration management
- Logging
- Basic Monitoring

The service proxies communicate using various protocols to the Oracle Communications software products used to deliver Convergence services.

Figure 1–1 Convergence Components

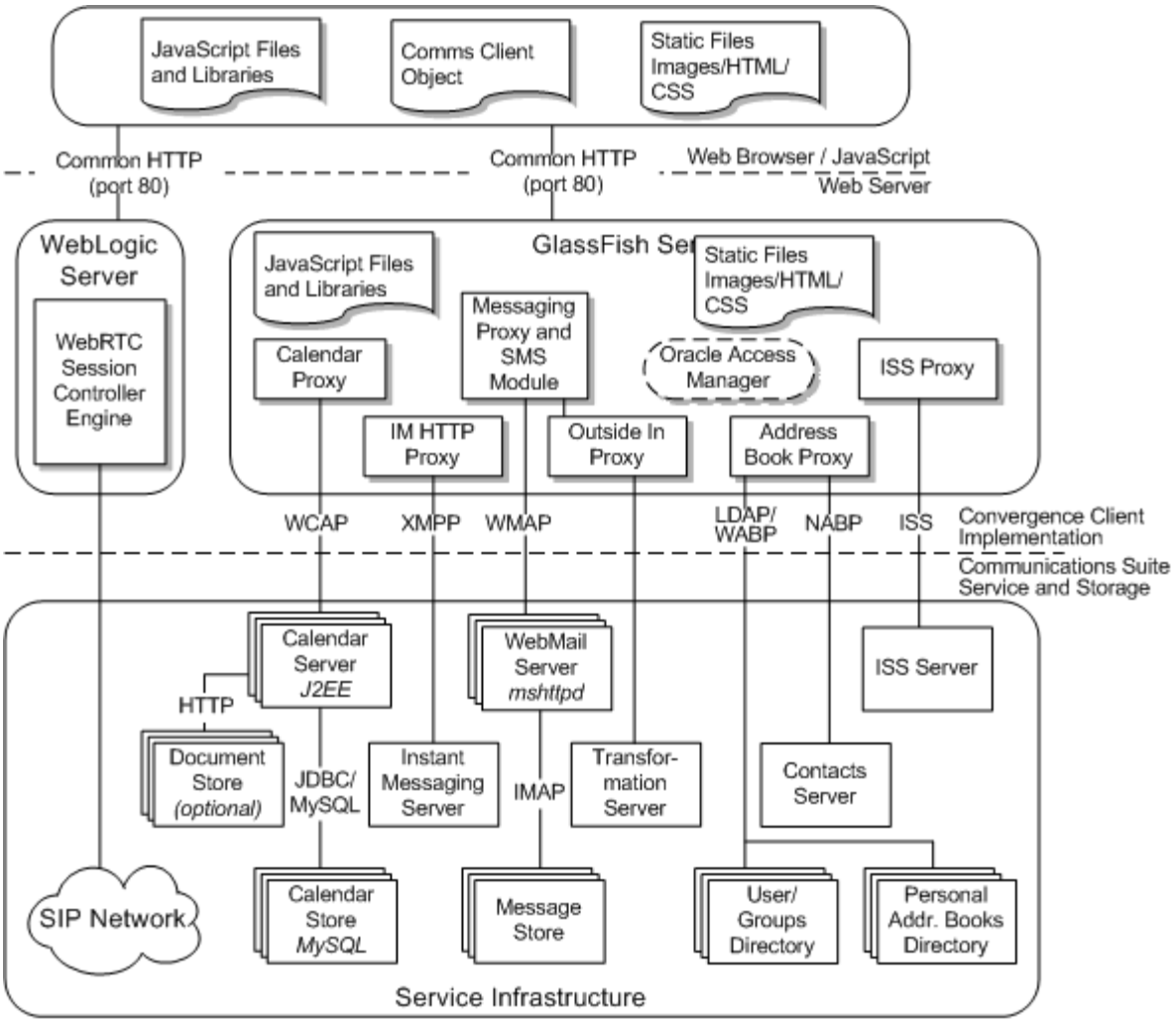


Recommended Deployment Topologies

Because Convergence is an end-user client program, it occupies the User Tier in any deployment topology.

Figure 1–2 shows the high-level logical Convergence architecture.

Figure 1–2 Convergence High-Level Logical Architecture



The general architectural recommendation is to use the well-known and generally accepted Internet-Firewall-DMZ-Firewall-Intranet architecture. For more information on addressing network infrastructure concerns, see the Unified Communications Suite wiki:

https://wikis.oracle.com/display/CommSuite/Determining+Your+Communications+Suite+Network+Infrastructure+Needs

Operating System Security

This section lists Convergence-specific OS security configurations. This section applies to all supported OSs.

Firewall Port Configuration

Convergence communicates with various components on specific ports. Depending on your deployment and use of a firewall, you might need to ensure that the firewalls are configured to manage traffic for the following components:

- GlassFish Server administration server (default 4848)
- Convergence (http 8080, https [default] 8181)

- WebMail Server (http 8990, https [default] 8991)
- Contacts Server (http 8080, https [default] 8181)
- Calendar Server (http 8080, https [default] 8181)
- Instant Messaging Server (default 5269, for both http and https)
- Indexing and Search Service (http 8080, https [default] 8181)
- LDAP (ldap 389, ldaps [default] 636)
- Oracle Access Manager (the WebLogic server default port)
- Outside In Transformation Server (default 60611)
- WebRTC Session Controller (the WebLogic server default port)

Close all unused ports, especially non-SSL ports. Opt for SSL-enabled ports, instead of non-SSL ports, for all communications.

For more information about securing your OS, see your OS documentation.

GlassFish Server Security

Convergence Server is deployed to a GlassFish server domain. For information about installing and configuring GlassFish Server, see *Oracle GlassFish Installation Guide*.

For information about securing GlassFish Server, see *Oracle GlassFish Security Guide*, at:

`http://docs.oracle.com/cd/E18930_01/html/821-2435/index.html`

Run the GlassFish Server installer in a Secure Administration Server Instance. If you do not run the GlassFish Server as an admin program in secure mode, then you are unable to run the Convergence init-config program in secure mode without running into errors. Therefore, install and configure the GlassFish server and Convergence in secure mode.

When installing GlassFish Server, you are asked for the following security information:

- Administration User and Administration User password
- master password for SSL certificate
- port number for HTTPS port
- secure administration server instance

Accessing a Web Application Deployed on GlassFish Server

To access a web application deployed on a GlassFish server, use the URL `http://localhost:8080/` (or `https://localhost:8181/` if it is a secure application), along with the context root specified for the web application. To access the GlassFish Server Administration Console, use the URL `https://localhost:4848/` or `http://localhost:4848/asadmin/` (its default context root).

Secure Sockets Layer (SSL)

Secure connections between applications connected over the Web can be obtained by using protocols such as Secure Socket Layer (SSL) or Transport Layer Security (TLS). SSL is often used to refer to either of these protocols or a combination of the two (SSL/TLS). Due to a security problem with SSLv3, Convergence recommends the use of only TLS. See "Disabling SSLv3 on Front-End GlassFish Server Hosts" for more information.

However, throughout this guide, secure communications may be referred to by the generic term SSL.

In a Convergence deployment, you can configure SSL between the following components:

- GlassFish Server administration server port
- GlassFish server for Convergence
- Messaging Server
- Contacts Server
- Calendar Server
- Instant Messaging Server
- Indexing and Search Service
- LDAP
- Access Manager
- Transformation Server
- WebRTC Session Controller

Configuring SSL in Convergence

SSL provides a secure means of communication between the web-browser client and the server.

You can enable SSL in Convergence when you run the Convergence configuration script the first time, or in GlassFish Server. If you are enabling SSL for Convergence in GlassFish Server, you must also set the **base.sslport** property using the Convergence **iwcadmin** command-line utility. For example:

```
iwcadmin -o base.sslport -v base_ssl_port
```

See *Convergence System Administrator's Guide* for more information about the **base.sslport** property and the **iwcadmin** command.

Configuring Authentication-Only SSL

Authentication-Only SSL is a mechanism in which users are authenticated by using the HTTPS protocol which sends user authentication details in an encrypted format. All other requests from the Convergence client are performed using the HTTP protocol. To configure Convergence to use Authentication-only SSL, you set the **base.sslport** and **base.enableauthonlyssl** properties using the **iwcadmin** command. For example:

```
iwcadmin -o base.sslport -v base_ssl_port
iwcadmin -o base.enableauthonlyssl -v true
```

See *Convergence System Administrator's Guide* for more information about the **base.sslport** and **base.enableauthonlyssl** properties and the **iwcadmin** command.

Enabling SSL for Back-End Servers

Using the **iwcadmin** command, you can enable a secure data connection between Convergence and the following back-end servers:

- To enable SSL to Messaging Server:

```
iwcadmin -o mail.enablessl -v true
iwcadmin -o mail.port -v mail_port
```

Messaging Server must be running in SSL mode.

- To enable SSL to Calendar Server 6.3:

```
iwcadmin -o cal.enablessl -v true
iwcadmin -o cal.port -v calendar_port
```

To enable SSL to Calendar Server 7:

```
iwcadmin -o caldav.enablessl -v true
iwcadmin -o caldav.port -v caldav_port
```

Calendar Server must be running in SSL mode.

- To enable SSL for Convergence address book, configure Convergence with SSL.
- To enable SSL between Convergence and Instant Messaging Server, you must enable TLS/SSL in Instant Messaging Server. No configuration changes are required for Convergence. See *Instant Messaging Server System Administrator's Guide* for more information.
- To enable SSL to Contacts Server:

```
iwcadmin -o nab.enablessl -v true
iwcadmin -o nab.port -v nab_port
```

- To enable SSL to Index Search Service:

```
iwcadmin -o ISS.enablessl  -v true
iwcadmin -o ISS.port  -v iss_port
```

- To enable SSL between Convergence and the directory server:

```
iwcadmin -o ugldap.enablessl -v true
iwcadmin -o ugldap.port -v ldap_port
```

Closing Non-SSL Connections

By default, Convergence listens to requests on both http (non-SSL) and https (SSL) connections. You should close all non-SSL connections, preventing Convergence from listening for non-SSL traffic.

To disable non-SSL connections:

1. List all the http listeners using the Oracle GlassFish Server **asadmin** command-line utility:

```
asadmin list  "*" | grep server-config  | grep http-listener
```

2. Determine which http listeners are open for non-SSL connections. Use the **asadmin** command to display all the settings for a particular http listener:

```
asadmin get server-config.http-service.http-listener.http_listener.*
```

Where *http_listener* is any http listeners returned by the **asadmin list** command.

3. For each non-SSL Convergence connection, use the **asadmin** command to disable the connection:

```
asadmin set
server-config.http-service.http-listener.http_listener.enabled=false
```

4. Restart the GlassFish server.

LDAP Security

To enhance client security in communicating with Directory Server, use a strong password policy for user authentication. For more information on securing Directory Server, see the discussion on security in *Oracle Directory Server Enterprise Edition Administration Guide*.

2

Performing a Secure Convergence Installation

This chapter presents planning information for installing Oracle Communications Convergence securely.

Installing Infrastructure Components Securely

Convergence is deployed within a GlassFish server domain. When installing and configuring Oracle GlassFish Server:

- Configure HTTPS and disable HTTP
- Configure the JMX Port for the GlassFish server to use SSL
- Configure the GlassFish server to prevent Denial of Service (DoS) attacks

To configure and administer GlassFish Server security, see *Oracle GlassFish Server Security Guide*.

Installing Third-Party Service Applications Securely

To provide its services, Convergence connects to other applications. The following applications must be installed securely.

- Convergence uses Oracle Communications Messaging Server to provide email services. See *Messaging Server Security Guide* for information about installing Messaging Server securely.
- Convergence can use Oracle Communications Contacts Server to provide address book services. See *Contacts Server Security Guide* for information about installing Contacts Server securely.
- Convergence uses Oracle Communications Instant Messaging Server to provide instant messaging services. See *Instant Messaging Server Security Guide* for information about installing Instant Messaging Server securely.
- Convergence uses Oracle Communications Indexing and Search Service to provide email attachment indexing and search services. See *Indexing and Search Service Security Guide* for information about installing Indexing and Search Service securely.
- Convergence uses Oracle Communications Calendar Server to provide calendar services. See *Calendar Server Security Guide* for information about installing Calendar Server securely.
- Convergence uses Oracle Outside In Technology to provide preview services of many common attachment types. See the Oracle WebCentre Content documentation for information about installing Outside In Technology securely.

- Convergence uses Oracle Communications WebRTC Session Controller to deliver real-time collaboration capabilities over the Web. See the WebRTC Session Controller documentation for information about performing a secure installation.

Credentials Needed to Install Convergence Components

If you are installing the Messaging Server webmail server component on the same host as Convergence, the Messaging Server installation asks for the following credentials:

- System user who will own the configuration files
- System group that will own the configuration files (of which system user is a part)
- Directory Server manager (bind DN and password)
- Messaging Server account passwords

The Convergence configuration program prompts for authentication credentials for the following:

- GlassFish Server administration server port number
- GlassFish administration user name and password
- User/Group Directory Server manager (bind DN and password)
- Webmail SSL port number
- Webmail Administration user ID and password
- Access in SSL mode between Messaging Server and Convergence
- Calendar Server SSL port number
- Calendar Administration user ID and password
- Access in SSL mode between Calendar Server and Convergence
- IM Httpbind component JID and password
- IM Avatar component JID and password
- Convergence administrator user name and password

3

Implementing Convergence Security

This chapter explains the security features for Oracle Communications Convergence.

Managing Security of Passwords

When using the **iwcadmin** command, you cannot include the `-W password_file` parameter unless the password file is encrypted. For this reason, the `-W` parameter is omitted from all examples in this guide.

Use the **iwcadmin** command to encrypt a password file:

```
iwcadmin -o admin.adminpwd
```

If you exclude the `-W password_file` parameter from your commands, the **iwcadmin** command prompts you for the administrator password.

See *Convergence System Administrator's Guide* for more information about the **iwcadmin** command.

Disabling SSLv3 on Front-End GlassFish Server Hosts

Identify the http-listener for the publicly accessible port that has SSL/TLS enabled (**security-enabled=true**) on which requests for Convergence are received. Ensure that SSLv3 is disabled for this listener by setting the option **ssl3-enabled** to **false**.

1. Identify the HTTP listeners that have SSL/TLS enabled (**security-enabled=true**) and verify whether SSLv3 is enabled on that listener (**ssl3-enabled=true**).

```
asadmin get configs.config.server-config.network-config.protocols.protocol.* |
grep http-listener.*security-enabled=true

configs.config.server-config.network-config.protocols.protocol.http-listener-2.
security-enabled=true

asadmin get
configs.config.server-config.network-config.protocols.protocol.http-listener-2.
ssl.ssl3-enabled

configs.config.server-config.network-config.protocols.protocol.http-listener-2.
ssl.ssl3-enabled=true
Command get executed successfully.
```

2. Disable SSL3 for those HTTP listeners.

```
asadmin set
configs.config.server-config.network-config.protocols.protocol.http-listener-2.
ssl.ssl3-enabled=false
```

```
configs.config.server-config.network-config.protocols.protocol.http-listener-2.
ssl.ssl3-enabled=false
Command set executed successfully.
```

3. Restart the GlassFish server.

Administering Encryption for Secure Authentication

Administering encryption for secure authentication includes:

- Configuring Convergence for SSL

  See "Secure Sockets Layer (SSL)" for more information.

- Setting up Certificate-based Authentication

  See "About Certificate-Based Authentication" for more information.

- Configuring Convergence for S/MIME

  See *Convergence System Administrator's Guide* for more information.

About Certificate-Based Authentication

This section explains key concepts about certificate-based authentication and how to set it up.

Enabling SSL and Client Authentication for a Listener in GlassFish Server

To enable SSL and client authentication for a listener in a GlassFish server:

1. Log into the GlassFish Server Administration Console.
2. Click on **Configurations**, then on **server-config**, then on **HTTP Service**, then on **HTTP Listener**.
3. Select a listener where SSL is enabled.

   The listener details appear.
4. Click the **SSL** tab.
5. Select the **Client Authentication Enabled** box.
6. Click **Save**.
7. Restart the GlassFish server.

Managing Certificates in GlassFish Server

Use the Java keytool for managing certificates. See your Oracle GlassFish Server documentation for more information.

Certificate Revocation Support in GlassFish Server

There are two types of Certificate Revocation Lists, static and dynamic. The following methods of certificate revocation are supported:

- Static Certificate Revocation List (CRL) checking
- Dynamic Certificate Revocation List (CRL) checking
- Revocation Checking using OCSP (Online Certificate Status Protocol)

See the Oracle GlassFish Server documentation for information about revocation checking.

Configuring the Convergence Certificate Mapper

A certificate mapper is used to map a certificate presented by a client to the user in the directory that should be associated with that certificate. Certificate mapping determines how Convergence server scans user entries in the LDAP directory. Use **certmap.conf** to configure how a client certificate is mapped to an LDAP entry. You can edit the **certmap.conf** file to add entries to match the structure of your LDAP directory and to list the certificates you want your users to have. Users can be authenticated based on attributes and their corresponding values used in the subjectDN.

The mapping file defines the following information:

- The LDAP tree location the server from where begins its search (Base DN)
- Certificate attributes the server should use as search criteria when searching for the entry in the LDAP directory
- Whether the server must go through an additional verification process

Mapper File Syntax

The file contains one or more named mappings, each applying to a different CA. Note that only lower-case entries can be used in <name>. A mapping has the following syntax:

```
certmap=<name1>,<name2>,<name3>...
<name1>.<property1>=<value>
<name1>.<property2>=<value>
```

For example:

```
certmap default,<name>,<name1>
default.<property1> =<value>

default.<property2> =<value>

<name>.<property1> =<value>
<name>.<property2> =<value>

<name1>.<property1> =<value>
<name1>.<property2> =<value>
```

The first line specifies a name(s) for the entry. You can use any name for the entry. If a mapper entry for a specific CA (as present in the IssuerDN in the client certificate) does not exist, the default configuration/mapping is used. Thus, it is recommended to configure default mappings.

Mapper File Properties and Values

The **certmap.conf** file has the following properties:

IssuerDN

The certificate authority IssuerDN of the certificates to which the map applies. (DN format as defined in RFC 2253). The IssuerDN must match the IssuerDN of the CA that issued the client certificate. For example, the following two IssuerDN lines have

different space separating the attributes, but the server treats these as two separate entries:

```
certmap sun1 ou=Sun Certificate Authority,o=Sun, c=US
certmap sun2 ou=Sun Certificate Authority, o=Sun, c=US
```

The following are the valid syntax for IssuerDN:

- empty - The corresponding map is invalid. The server will fall back to use the default map. (For example: usps.IssuerDN=).
- commented out/not set - The corresponding map is invalid. The server will fall back to use the default map.
- Valid RFC 2253 DN - This map will be used for all the certificates, whose IssuerDN matches this DN.

DNComps

The DNComps property is used to configure the LDAP tree location from where the server begins its search (Base DN).

Specifies a comma separated list of relative distinguished name components of the base DN for an LDAP search to find the user entry matching the certificate. The components are taken from the subject DN of the certificate.

The server gathers values for these attributes from the client certificate and uses the values to form an LDAP DN, which determines where the server starts its search in the LDAP directory. For example, if you set DNComps to use the o and c attributes of the DN, the server starts the search from the o=<org>, c=<country> entry in the LDAP directory, where <org> and <country> are replaced with values from the DN in the certificate.

Any attribute for which a mapping is defined but is not contained in the certificate subject is not included in the generated base DN.

The following are the valid syntax for DNComps:

- empty - The server performs a sub-tree search with base DN as the configured Base DN (dcroot) in convergence configuration. (For example: default.DNComps=)
- commented out - take the existing user's DN from the cert and this becomes the Base DN. (For example: #default.DNComps=ou,o)
- attribute names - a comma separated list of attributes to form DN. The attribute values are taken from the subject DN of the certificate.

  Attribute substitutions - attribute and their corresponding LDAP attribute to be used as a substitute while constructing the Base DN. The attribute values are taken from the subject DN of the certificate. (For example: default.DNComps=ou=OrgUnit,o).

  Table 3–1 lists the supported attributes for this mapper (as per RFC 2253 and RFC 5280).

Table 3–1 Supported attributes for DNComps

Attribute Keyword	Description
cn	Common name
l	Location
street	Street

Table 3–1 (Cont.) Supported attributes for DNComps

Attribute Keyword	Description
ou	Organization unit
o	Organization
c	Country
uid	Unique ID
emailaddress	Email address (RFC 822)
s	State
serialsnumber	Serial number for a name
dnq/dnqqualifier	DN specific information
t	Person's title (rank)
surname	Last name
givenname	First name
initials	Initials
generation	Example, jr, III
ip	IP address

FilterComps

FilterComps specifies a comma separated list of attributes to form a filter for an LDAP search to find the user entry matching the certificate. The values for the filter are taken from the Subject DN of the client certificate.

If multiple attribute mappings are defined, then the server combines them with an AND search. For example, if two mappings are defined cn and uid, and the server is presented with a certificate having a subject of UID=john.doe@example.com,CN=John Doe,O=Example Corp,C=US, then it generates a search filter of (&(cn=John Doe)(uid=john.doe@example.com)).

Any attribute for which a mapping is defined but is not contained in the certificate subject is not included in the generated search filter. All attributes that can be used in generated search filters should have corresponding indexes in all back-end databases that can be searched by this certificate mapper. For the mapping to be successful, the generated search filter must match one user in the directory (within the scope of the base DNs for the mapper). If no users or multiple users match the generated criteria, the mapping fails.

The following are the valid syntax for FilterComps:

- empty - The generated search filter is (objectclass=*) (For example: default.FilterComps=)
- commented out - The generated search filter is (objectclass=*) (For example: #default.DNComps=cn)
- attribute names - a comma separated list of attributes to form search. The attribute values are taken from the subject DN of the certificate. If multiple attribute mappings are defined, then the server combines them with an AND search.

  Attribute substitutions - attribute and their corresponding LDAP attribute to be used as a substitute while constructing the search filter. The attribute values are taken from the subject DN of the certificate. (For example: default.FilterComps=uid=employeeID,emailaddress=mail)

Table 3–2 lists the supported attributes for this mapper (as per RFC 2253 and RFC 5280).

Table 3–2 Supported attributes for FilterComps

Attribute Keyword	Description
cn	Common name
l	Location
street	Street
ou	Organization unit
o	Organization
c	Country
uid	Unique ID
emailaddress	Email address (RFC 822)
s	State
serialsnumber	Serial number for a name
dnq/dnqqualifier	DN specific information
t	Person's title (rank)
surname	Last name
givenname	First name
initials	Initials
generation	Example, jr, III
ip	IP address

CmapLdapAttr

CmapLdapAttr specifies the name of the LDAP attribute in the directory containing the subject DN of the certificate.

Unless this attribute is empty, the baseDN is the convergence configured Base DN (DC root) with a search filter containing subject DN. For the mapping to be successful, the certificate mapper must match exactly one user (within the scope of the base DNs for the mapper). If no entries or multiple entries match, the server performs a search with the baseDN generated from the corresponding DNComps and the search filter from the corresponding FilterComps.

The following is the valid syntax for CmapLdapAttr:

- Attribute name - is a name for the attribute in the LDAP directory that contains subject DNs from all certificates belonging to the user. (For example: default.CmapLdapAttr=certSubjectDN)

VerifyCert

VerifyCert tells the server whether it should compare the client's certificate with the certificate found in the LDAP directory. It takes two values: on and off. This feature is useful to ensure your end-users have a valid, un-revoked certificate. However, you should only use this property if your LDAP directory contains certificates.

The default behavior is the same as off, meaning client certificates are not checked to be valid and not revoked.

The following is the valid syntax for VerifyCert:

- on - compare's client certificate with the certificate found in the userCertificate attribute in user's LDAP entry (For example: default.VerifyCert=on)

Sample certmap.conf Mapping

The **certmap.conf** file should have at least one entry. The following examples illustrate the different ways you can use the **certmap.conf** file.

The following example represents a **certmap.conf** file with only one default mapping:

```
certmap=default
default.DNComps=ou, o, c
default.FilterComps= emailaddress=mail, uid
default:verifycert=on
default.issuerDN=default
```

Using this example, the server starts its search at the LDAP branch point containing the entry {{ou=<orgunit>, o=<org>, c=<country>}} where the text within <> is replaced with the values from the subject's DN in the client certificate.

The server then uses the values for email address and user ID from the certificate to search for a match in the LDAP directory. The search filter would be &(mail=<email>)(uid=<uid>)). When it finds an entry, the server verifies the certificate by comparing the one the client sent to the one stored in the directory.

The following example file has two mappings, one for default and another for the US Postal Service:

```
certmap=default,usps
default.DNComps=
default.FilterComps=emailaddress, uid
usps.IssuerDN=ou=United States Postal Service, o=usps, c=US
usps.DNComps=ou,o,c
usps.FilterComps=emailaddress=mail
usps.verifycert=on
default.issuerDN=default
```

When the server gets a certificate from someone other than the USPS, it uses the default mapping, which starts at the configured base DN (DC root) and searches for an entry matching the client's user ID and email address. If the certificate is from the USPS, the server starts its search at the base DN containing the organizational unit and searches for matching email addresses. Note that if the certificate is from the USPS, the server verifies the certificate; other certificates are not verified.

> **Note:** The IssuerDN in the certificate must be identical to the IssuerDN listed in the first line of the mapping. In the previous example, a certificate from an issuer DN that is o=United States Postal Service,c=US will not match because there is no space between the o and the c attributes.

The following example uses CmapLdapAttr to scan the LDAP database for the certSubjectDN attribute whose value exactly matches the entire subject DN taken from the client certificate.

```
certmap=myco
myco.IssuerDN=ou=My Company Inc, o=myco, c=US
myco.CmapLdapAttr=certSubjectDN
```

```
myco.DNComps=o, c
myco.FilterComps=emailaddress=mail, uid
myco.verifycert=on
```

If the client certificate subject is:

```
uid=Walt Whitman, o=LeavesOfGrass Inc, c=US
```

the server first searches for entries that contain the following information:

```
certSubjectDN=uid=Walt Whitman, o=LeavesOfGrass Inc, c=US
```

If a matching entry is found, the server verifies the entry. If no matching entries are found, the server uses DNComps and FilterComps to search for matching entries. In this example, the server would search for uid=Walt Whitman in all entries under o=LeavesOfGrass Inc, c=US.

Enabling Certificate Authentication Support

To enable certificate authentication support using the **iwcadmin** command, set the **auth.cert.enable** property to **true**. Then restart the GlassFish server.

Note: In GlassFish Server, if certificate authentication support is enabled when a user accesses Convergence without certificate authentication, then certificate authentication support will fall back to form-based authentication. To ensure proper fallback, add the <property name="com.sun.grizzly.ssl.auth" value="want"/> to the http-listener on your GlassFish server.

If the form-based option is set to true, and certificate-based authentication is disabled, when the user accesses Convergence without certificate authentication, a login page is displayed. Additionally, Convergence can be accessed on another port.

To enable fallback to form-based authentication when certification authentication is enabled, set **auth.cert.enablefallback** to **true** using the **iwcadmin** command.

About Single Sign-On Security

You can enhance Convergence security with single sign-on (SSO).

Oracle recommends that you deliver SSO functionality using Oracle Access Manager. You can also use Trusted Circle SSO.

See *Convergence System Administrator's Guide* for information about enabling SSO.

You can write your own customized SSO module for Convergence. See "Writing a Custom Pluggable SSO Module" for more information. The SSO module is not an SSO provider, but rather enables an entire site with the same SSO provider to use SSO between Convergence and other web applications.

About Mail Encryption and Digital Signatures

You can configure Convergence to enable email encryption and digital signing. Convergence uses S/MIME to encrypt and sign email messages.

See *Convergence System Administrator's Guide* for information about enabling S/MIME for email encryption.

Inform and educate users about sending encrypted email messages. The Convergence online Help explains how to encrypt email messages.

Detecting Security Attacks or Insecure System Use

The following sections can help monitor Convergence for security threats and prevent possible attacks.

Messaging Server Best Practices for Fighting Email Spam

Email spam detection and prevention is a complex issue that is difficult to eliminate. You can reduce its impact on your system, internal users, and external email recipients.

See *Messaging Server Security Guide* for more information.

Limiting Mail Delivery

Messaging Server provides a number of mechanisms to limit or throttle email delivery. These mechanisms can help prevent intentional or accidental Denial-of-Service attacks.

See *Messaging Server Security Guide* for more information.

Denying Denial of Service Attacks in Messaging Server

The following article on the Oracle technology web site explains several ways you can configure Messaging Server to prevent denial of service (DoS) attacks:

`http://www.oracle.com/technetwork/systems/articles/preventingdosattacks-jsp-138891.html`

System Logging

Repeated login failures could be indicative of an external party trying to gain access to an account. You can review LDAP server logs for failed logon attempts. With Oracle Internet Directory, logon failures are recorded as Bind Failures in the access log file **access.txt**. See your LDAP or directory server documentation for more information about logging.

Securing Oracle Outside In Transformation Server

You can install Oracle Outside Transformation Server In to enable Convergence to preview all kinds of attachments in the browser. Oracle Outside In is an optional module.

The Convergence server authenticates users before they can view attachments. Convergence generates a session cookie that contains a server-generated URL for each attachment request to the Outside In proxy. The proxy sends the request to the Transformation Server, which renders the attachment for the user to view.

The server generated URL is not displayed in Convergence.

If you integrate Convergence with Oracle Outside In Transformation Server, you must configure the Outside In Transformation Server to prevent security attacks.

- Configure a time out to purge files stored in the transformation server. By default, the time out is set to 5 minutes.
- Configure each user directory to store a maximum number of files before replacing existing older files. By default, each user directory can hold a maximum of 10 files.

See the discussion about managing attachment life cycles in *Convergence System Administrator's Guide* for more information.

By default, Convergence imposes a maximum size on outgoing email attachments of 5MB. Depending on how Oracle Outside In Transformation Server is set up, you may need to adjust the maximum attachment size to decrease the risk of overloading the transformation server.

4

Security Considerations for Developers

This chapter explains how you can write custom modules that enhance security for Oracle Communications Convergence.

Writing a Custom Pluggable SSO Module

This information describes how to write a pluggable custom Single Sign-On (SSO) module for Convergence. Convergence offers two Single Sign-on mechanisms:

- Access Manager SSO (Legacy Mode and Realm mode).
- Messaging SSO (also referred to as Trusted Circle SSO).

By itself, the pluggable custom SSO module is not an SSO provider nor is it a replacement for any identity or access management services. Instead, the pluggable custom SSO module allows a site to use SSO between Convergence and another web application, where they all use the same SSO provider for identity or access management.

If you want your deployment to use a different SSO mechanism, you must write and implement an SSO module. Internally, Convergence uses a proxy-auth mechanism to perform SSO with Oracle Communications back-end servers. The back-end servers are: Messaging Server, Calendar Server, and Instant Messaging. Convergence enables you to write custom SSO modules to provide Single Sign-On.

SSO Mechanism in Convergence

As with any SSO-aware application, when a user is authenticated by using Access Manager for example, Convergence loads the authentication module to validate the user. On successful validation, the user is allowed to access the application. If the validation is not successful, the user is redirected to the login page.

Implementing the Custom SSO Module

Before designing a solution for the custom SSO module, Convergence SSO provider framework needs to be implemented:

- All custom SSO modules must implement SSOProvider interface.
- Convergence uses LDAP (Schema 1 or Schema 2) to store user data. This is called UG LDAP.
- UG LDAP uses the LDAP filter to identity user in UG LDAP.
- The SSO provider must provide the UG LDAP user identifier and domain identifier.

- After SSO validation the implementation must provide valid *uid* and valid domain/organization of the user in UG LDAP. This information will be obtained by SSO framework by invoking *getUid()* and *getDomain()* method of custom SSO Provider.
- The SSO implementation can use any other class that requires custom SSO module to work.

To write a custom SSO module:

1. Convergence defines a set of interfaces and class that need to be implemented. They are:
   - **SSOProvider.java**
   - **SSOListener.java**

     SSOProvider and SSOListener interfaces have to be implemented by the same class.
2. Configure the SSO module using the **iwcadmin** command.

The following example shows the reference implementation for **SSOProvider.java**:

```
package com.sun.comms.client.security.sso;

import java.util.Properties;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
* Custom SSO provider must implement this interface.
*/
public interface SSOProvider {
    /**
     *  SSO framework in Convergence will invoke this method by passing all
required SSO configuration, that are configured in configuration.
     *  Implementation can store this info for future use. keys in initConfig are
case sensitive.
     */
    public void init(Properties initConfig);

    /**
     * This method will be invoked by SSO framework after calling init() method.
Implementation can have SSO validation code here.
     * If SSO validation or Single-Sign-On is successful, this method should
return true.
     * If SSO validation succeeds implementation must not create http session
here. It is taken care by SSO framework
     */
    public boolean SingleSignOn(HttpServletRequest request,HttpServletResponse
response) throws SingleSignOnException;

    /**
     * This method will be invoked by SSO framework when user logs out of
application and Single-Sign-Off is enabled in configuration.
     * If SSO validation or Single-Sign-Off is successful, this method should
return true.
     * Implementation can perform SSO provider specific Single-Sign-Off here like
cleanup SSO cookies in response.
     */
    public boolean SingleSignOff(HttpServletRequest request,HttpServletResponse
response) throws SingleSignOffException;
```

```
    /**
     * This method will be called by SSO framework if Single-Sign-On succeeds.
Implementation must provide a uid (user identifier) of the user
     * in UG LDAP. This will be used by framework to load authenticated user from
UG LDAP.
     */
    public String getUid();

    /**
     * This method will be called by SSO framework if Single-Sign-On succeeds.
Implementation must provide a domain/organization (domain identifier) of the user
     * in UG LDAP. Framework will use this to locate the user under this domain in
UG LDAP.
     */
    public String getDomain();

    /**
     * How much more time SSO token is valid with SSO Provider. Currently not used
by framework and hence can be ignored.
     */
    public long getTimeLeft();

}
```

The following example shows the reference implementation for **SSOListener.java**:

```
package com.sun.comms.client.security.sso;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/*
* If SSO provider needs to perform some post Single-Sign-On operation. This
interface must be implemented.
*/
public interface SSOListener {
    /**
     * This method will be invoked by framework if Single-Sign-On operation
succeeded, user entry is loaded and http session is created.
     * Implementation can do postSignOn related tasks here e.g. registering token
listener for sso token notification etc.
     *
     * @param request - Http request for single sign on
     * @param response - Http response for single sign on
     * @param session- Convergence session that is created after successful single
sign on
     */
    public void postSignOn(HttpServletRequest request,HttpServletResponse
response,HttpSession session);

}
```

SingleSignOffException - Exception thrown if SingleSignOff fails for any abnormal condition.

SingleSignOnException - Exception thrown if SingleSignOn fails for any abnormal condition.

Note: While implementing the custom SSO module, **iwc.jar** should be available in the classpath of development environment. The **iwc.jar** file requires SSO module classes.

Configuration

Once the required classes for the SSO module are created, you must configure it to work with Convergence server. To configure the SSO module, perform the following operations:

1. Configure the SSO module using the **iwcadmin** command:

```
iwcadmin -o sso.enable -v "true"
iwcadmin -o sso.enablesignoff -v "true"
iwcadmin -o sso.servicename -v CUSTOM_SSO
iwcadmin -o sso.ssoserviceimpl -v "com.client.sample.CustomSSOProvider"
iwcadmin -o sso.misc.config_name1 -v "Config_Val1"
iwcadmin -o sso.misc.cofnig_name2 -v "Config_Val2"
```

Note: All the miscellaneous configuration parameters such as *config_name1* and *config_name2* and their values are case sensitive. These parameters should match the in the *SSOProvider classes' init*() method.

2. Create a JAR file with all custom classes and supporting classes.
3. Copy the JAR file to the Convergence *Convergence_Domain***/applications/Convergence/WEB-INF/lib** directory.

About the *sso.notifyserviceimpl* Parameter

In addition, you might choose to enable the *sso.notifyserviceimpl* parameter, which can be any user defined class that can listen to events from an SSO provider such as Access Manager. The class name is available through configuration properties passed to the custom SSOProvider implementation class (for example: *NotificationServiceImplementation* as key). In a custom SSO Provider implementation, you can obtain the class name, create the object, and register it as a listener for SSO events such as token expiration, single sign off notification, and so forth. This implementation is an SSO Provider specific class like AMSDK; it is different from SSOListener.

Custom SSO Implementation Example

The following example (**CustomSSOProvider.java**) shows a custom SSO reference implementation:

```
**** BEGIN com/client/sample/CustomSSOProvider.java ****

package com.client.sample;

import com.sun.comms.client.logging.IwcLogger;
import com.sun.comms.client.security.sso.SSOProvider;
import com.sun.comms.client.security.sso.SSOListener;
import com.sun.comms.client.security.sso.RenewSSO;
import com.sun.comms.client.security.sso.SingleSignOffException;
import com.sun.comms.client.security.sso.SingleSignOnException;
```

```
import com.sun.comms.client.security.sso.GeneralSSOException;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import java.util.Properties;
import org.apache.commons.logging.Log;

public class CustomSSOProvider
implements SSOProvider,SSOListener,RenewSSO
{

  private static final Log logger = IwcLogger.getLogger(IwcLogger.AUTH_LOGGER);

  public CustomSSOProvider() {
    logger.debug("Custom SSO Provider created");
  }

  public void init(Properties props)
  {
    logger.debug("init() called");
  }

  public String getDomain()
  {
    logger.debug("getDomain() called");
    return "domain.com";
  }

  public long getTimeLeft()
  {
    logger.debug("getTimeLeft() called");
    return 3600;
  }

  public String getUid()
  {
    logger.debug("getUid() called");
    return "uid";
  }

  public void refreshSSO(HttpServletRequest request,HttpServletResponse response)
    throws GeneralSSOException {
    logger.debug("refreshSSO() called");
  }

  public boolean SingleSignOn(HttpServletRequest request, HttpServletResponse
response)
    throws SingleSignOnException
  {
    logger.debug("SingleSignOn() called");
    return true;
  }

  public void postSignOn(HttpServletRequest request, HttpServletResponse response,
HttpSession session) {
    String sessionid = session.getId();
    String token = (String) session.getAttribute("USER_TOKEN");
    String cookieValue = "jid=" + sessionid + ":token=" + token;;
    logger.debug("postSignOn() called - create a new cookie SSOIwcAuth=" +
```

```
cookieValue);

    Cookie cookie = new Cookie("SSOIwcAuth", cookieValue);
    cookie.setPath("/");
    cookie.setDomain(".example.com");
    response.addCookie(cookie);
  }

  public boolean SingleSignOff(HttpServletRequest request, HttpServletResponse
response)
    throws SingleSignOffException
  {
    logger.debug("SingleSignOff() called");
    return true;
  }
}

**** END com/client/sample/CustomSSOProvider.java ****

[/tmp/test]$ cat compile.sh
#!/usr/bin/bash

echo "Compiling...."
/usr/jdk/instances/jdk1.6.0/bin/javac -classpath \
/opt/sun/comms/iwc/web-src/server/WEB-INF/lib/commons-logging-1.1.jar:/opt/sun/
comms/iwc/web-src/server/WEB-INF/lib/iwc.jar:/opt/SUNWappserver/lib/javaee.jar \
com/client/sample/CustomSSOProvider.java

echo "Creating JAR file"
/usr/jdk/instances/jdk1.6.0/bin/jar -cvf customsso.jar \
com/client/sample/CustomSSOProvider.class
```

Summary

Convergence enables you to write your own custom SSO authentication modules. To write a custom SSO module, the Convergence SSO framework requires that you implement the following interfaces:

- *SSOProvider*
- *SSOListener*

Additionally, you can also use other classes that help you to implement the SSO module. Finally, you must configure Convergence to use the custom SSO module that you created using the **iwcadmin** command.

Writing a Custom Authentication Module

Convergence server provides an interface that enables you to create custom user authentication in the form of a customizable Java-based authentication module. The custom authentication module allows an organization to use a non-Oracle LDAP mechanism (for example, an RDBMS, flat-text file or third-party LDAP server) to provide authentication functionality.

By default, Convergence uses Oracle Directory Server Enterprise Edition for authentication store.

Basic Concepts

This section defines the terms and describes the authentication framework architecture and its components.

Convergence uses the following repositories to store user data:

- **User Authentication Store**: Contains user credentials, such as user name, password, domain information, and a unique identifier to identify the user in the User or Group LDAP store.
- **User/Group LDAP Store (UG LDAP)**: Contains user preferences such as time zone that the user is in, language preference, and user theme. Convergence uses Schema 1 or Schema 2 to store user information in the User or Group LDAP.

Convergence Authentication Framework

This section describes how the authentication framework works in Convergence.

1. The authentication module first authenticates the user in the authentication store using the configured authentication module. The default authentication module that works by default is Oracle Directory Server Enterprise Edition.
2. On successful authentication, the authentication module gets the user specific attributes like user ID, the domain of the user, organization, and a unique identifier.
3. The authentication framework loads the user from the User Group LDAP using the user ID (**userID**) and domain name (**userDomain**).

Contracts Defined by the Authentication Module

Before designing a solution for the custom authentication module, you must be aware of the contracts that the Convergence authentication framework needs for successful transfer of information between the authentication store, the Convergence authentication framework, and UG LDAP.

- The authentication module must provide a mechanism to identify a user in the UG LDAP after successful authentication. The custom authentication can have any authentication store that can use any type of identifier to authenticate the user. The authentication mechanism should provide a relationship between the authenticated user and UG LDAP. After successful authentication, the authentication module should provide a unique identifier to locate the user in the UG LDAP. For example, if both authentication store and UG LDAP use the same identifier to locate the user, after successful authentication, the authentication module must set **userID** and **userDomain** parameters in the HTTP request by using callback handler objects. These parameters are used by UG LDAP filter to load the user from the UG LDAP. In our example, the user ID (example *scott*) is the unique identifier used to locate the user in UG LDAP.
- All the custom authentication modules must implement the following three classes:
  - JAAS **LoginModule** interface. Convergence uses JAAS **LoginModule** as an interface for all its login modules. The custom authentication module must implement this interface. Although the authentication module uses the JAAS framework for authentication, it does not use all the advanced capabilities like authentication chaining, and multiple login modules.
  - **HttpCallbackHandler**. An abstract class that implements the **CallbackHandler** of JAAS. This class must be implemented to handle custom

callbacks. All custom authentication modules must implement this class to handle custom callbacks.

  - Convergence uses the JAAS **LoginCallback** and **CallbackHandler** interface to get and set information between the authentication module and Convergence application. Since Convergence is a web application, authentication is performed through HTTP based request and response. Convergence provides an abstract class: **HttpCallbackHandler**, which implements **CallbackHandler** interface of JAAS.

- After successful authentication, the authentication module must set the **UserPrincipal** object in the **Subject**. This can be done in commit method of login module. **UserPrincipal** must be created using **loginID** of the user.
- Custom authentication modules must not create HTTP session (**HTTPSession**) object. Convergence authentication framework takes care of initializing the session.

About the Sample Application

This section describes the various files that are created for the custom authentication module to work. Use this as a reference to create other custom authentication modules to suit your enterprise' needs. The sample authentication module can be used as is by copying the source files and following the steps as mentioned in the following sections.

Caution: If you must change the core class file names provided in this section, you must appropriately refractor the code. Some files use objects created by other core classes of the custom authentication module.

This example describes an authentication module for a file based user data store. The following is a sample set of data that could be used to store user information in the data store.

userinfo.txt

```
smith:test:siroe.com
jack:test:siroe.com
scott:test123:siroe.com
```

In the example, each attribute is separated by a colon. For example, the first record of the file provides information about the user ID *smith* whose password is *test* with domain **siroe.com**.

Implementing the Classes Required for the File-Based Authentication Store

This section describes the classes that are used to implement the authentication module for a file-based user store. The following are the core class:

- **SunTestLoginModule.java**

```
package com.sun.comms.test;

import com.sun.comms.client.logging.IwcLogger;
import com.sun.comms.client.security.auth.UserPrincipal;
import java.io.BufferedReader;
import java.io.File;
```

```
import java.io.FileReader;
import java.io.IOException;
import java.util.Map;
import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.FailedLoginException;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;
import org.apache.commons.logging.Log;
import com.sun.comms.test.SunTestAuthCallBack;

public class SunTestLoginModule implements LoginModule {

    private Subject subject;
    private CallbackHandler cbh;
    private Map sharedState;
    private Map options;
    private boolean succeeded;
    private UserPrincipal up;
    private SunTestAuthCallBack mcb = null;
    private String credFile = "";
    private final static Log logger =
IwcLogger.getLogger(IwcLogger.AUTH_LOGGER);

    public void initialize(Subject subject, CallbackHandler callbackHandler,
Map<String, ?> sharedState, Map<String, ?> options) {
        this.subject = subject;
        this.cbh = callbackHandler;
        this.sharedState = sharedState;
        this.options = options;
        credFile = (String) options.get("CredentialFile");
    }

    public boolean login() throws LoginException {
        Callback[] callbacks = new Callback[1];
        mcb = new SunTestAuthCallBack();
        callbacks[0] = mcb;

        if (cbh == null) {
            throw new LoginException("Error: no CallbackHandler available " +
                    "to gather authentication information from the user");
        }

        try {
            // Get userid and pwd from request
            cbh.handle(callbacks);
        } catch (Exception ex) {
            throw new LoginException("SunTestLoginModule: login failed");
        }

        String[] userInfo = attemptLogin();

        if (userInfo != null && userInfo.length==3) {
            mcb.setUserInfo(userInfo[0], userInfo[2]);
            succeeded = true;
            return true;
        } else {
            System.err.println("Unable to find user entry");
            throw new FailedLoginException("Unable to find user entry");
```

```
        }
    }

    private String[] attemptLogin() throws LoginException {

        if(credFile==null)
            throw new LoginException("User database file is not set
configuration.");

        File loginFile = null;
        String userID = mcb.getUserName();
        String userPwd = mcb.getUserPwd();

        if (userID == null || userPwd == null) {
            throw new LoginException("Required user credential not found");
        }

        try {
            loginFile = new File(credFile);
            if (loginFile.exists()) {

                BufferedReader reader = new BufferedReader(new
FileReader(loginFile));
                String userEntry = null;
                while ((userEntry = reader.readLine()) != null) {
                    String[] usrAcc = userEntry.split(":");
                    if (usrAcc != null && usrAcc.length == 3) {
                        if (userID.equals(usrAcc[0]) &&
userPwd.equals(usrAcc[1])) {
                            return usrAcc;
                        }
                    }
                }

            } else {
                System.err.println("Unable to find user database file " +
credFile);
                throw new LoginException("Unable to find user database file " +
credFile);
            }
        } catch (IOException ex) {
            System.err.println("Unable to load user database file " +
credFile);
            throw new LoginException("Unable to load user database file " +
credFile);
        }
        return null;
    }

    public boolean commit() throws LoginException {
        if (succeeded == false) {
            return false;
        } else {
            // add a Principal (authenticated identity) to the Subject
            UserPrincipal userPrincipal = new UserPrincipal(mcb.getUserName());

            if (!subject.getPrincipals().contains(userPrincipal)) {
                subject.getPrincipals().add(userPrincipal);
            }
        }
```

```
        return true;
    }

    public boolean abort() throws LoginException {
        return true;
    }

    public boolean logout() throws LoginException {
        return true;
    }
}
```

- **SunTestAuthCallBack.java**

```
package com.sun.comms.test;

import com.sun.comms.client.security.auth.LoginCallback;
import java.io.Serializable;
import java.net.InetAddress;
import java.net.UnknownHostException;
import java.util.Locale;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class SunTestAuthCallBack implements LoginCallback, Serializable {

    HttpServletRequest req;
    HttpServletResponse res;
    String username = null;
    String pwd = null;

    protected String name = null;
    protected String host = null;
    protected String user = null;
    protected String userDomain = null;
    protected Locale locale = null;
    protected String serverName = null;

    SunTestAuthCallBack(){

    }

    public void setData(HttpServletRequest request,HttpServletResponse
response){
        this.req = request;
        this.res = response;
        username = (String)req.getParameter("username");
        pwd = (String)req.getParameter("password");


    }

    public String getUserName(){
        return username;
    }

    public String getUserPwd(){
        return pwd;
    }
```

```
    public void setUserInfo(String uid,String domain){
        req.setAttribute("loginID", uid);
        req.setAttribute("userDomain", domain);
    }

    public boolean setData(Object obj) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    public Locale getLocale() {

      if (locale == null)
          return Locale.getDefault();

        return locale;
    }

    /**
     * set the client locale
     */

    public void setLocale(Locale locale) {
        if (locale != null)
            this.locale = locale;
    }

    /**
     * get the host name of the machine running the console.
     * this may be required for auditing purposes
     */

    public String getHost() {

    if (host == null) {
        try {
                host = InetAddress.getLocalHost().getHostName();
            } catch (UnknownHostException ukhe) {
        host = null;
            }
    }

        return host;
    }

    /**
     * set the host name of the machine
     */

    public void setHost(String host) {
        if (host != null)
            this.host = host;
    }

    @Override
    public void loadData() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

- **AppTestCallbackHandler.java**

```
package com.sun.comms.test;

import com.sun.comms.client.logging.IwcLogger;
import com.sun.comms.client.security.auth.modules.HttpCallbackHandler;
import com.sun.comms.client.web.RequestContextProvider;
import java.io.IOException;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.UnsupportedCallbackException;
import org.apache.commons.logging.Log;
import com.sun.comms.test.SunTestAuthCallBack;


public class AppTestCallbackHandler extends HttpCallbackHandler {
    private final static Log logger =
IwcLogger.getLogger(IwcLogger.AUTH_LOGGER);
    public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
        if (callbacks == null) {
        throw new IOException("Empty or null callback array");
    }

    for (int i = 0; i < callbacks.length; i++) {
              if (callbacks[i] instanceof SunTestAuthCallBack ) {
                 SunTestAuthCallBack nc = (SunTestAuthCallBack)callbacks[i];
                 nc.setData( RequestContextProvider.getHttpRequest(),
RequestContextProvider.getHttpResponse());
                 System.err.println("request and response set in
AppTestCallbackHandler");
             }else
                 System.err.println("Callback objects are not instance of
SunTestAuthCallBack");
        }
    }
}
```

How the Implementation Works

For every authentication request, the Convergence authentication framework reads the configured login module class name, call back handler class name and executes it using JAAS framework.

The JAAS framework calls the **initialize()** method by passing all the required arguments. One important argument is the Map option of the **initialize()** method. Convergence's authentication framework populates this object with all Misc parameters of **CustomJAASService** configuration.

In this example, we pass the directory location of user database **CredentialFile** as part of Misc parameter to **SunTestLoginModule**. The other arguments are:

- **Subject subject** - represents Subject that is being authenticated.
- **CallbackHandler callbackHandler** - Object that is responsible for handling custom callbacks.
- **Map sharedState** - Not used by Convergence and hence ignore it.

After successful initialization, the login module obtains all the required information about the callback handler and all the required configuration. The JAAS framework then invokes the **login()** method. This method performs the authentication, which is module specific. In this sample, **login()** method first creates callback object(s):

```
Callback[] callbacks = new Callback[1];
mcb = new SunTestAuthCallBack();
```

The call back object is aware of how to obtain the authentication related information such as the **username**, **password**, and so on. This is returned as a HTTP request. Once call back objects are created, it passes callback objects to **CallBackHandler**'s handle method.

```
cbh.handle(callbacks);
```

callbackhandler knows how to handle call back objects. For example, the method used for callback object, the data to be passed to it, and so on.

the **handle()** method of callback handler then calls callback object's **setData()** by passing request and response objects:

```
SunTestAuthCallBack nc = (SunTestAuthCallBack)callbacks[i];
nc.setData(
            RequestContextProvider.getHttpRequest(),
            RequestContextProvider.getHttpResponse());
```

Now, the Callback's **setData()** extracts the required information from request and response. In this sample, it gets request parameter **username** and **password** from request.

```
this.req = request;
    this.res = response;
    username = (String)req.getParameter("username");
    pwd = (String)req.getParameter("password");
```

The callback object now has all information that is required to authentication the user from the HTTP request. Now login **method()** calls an internal method **attemptLogin()**. This method obtains login information from the callback object:

```
String userID = mcb.getUserName();
        String userPwd = mcb.getUserPwd();
```

and loads the user database file and performs authentication. If authentication is successful this method returns **String** array with **userID** and **userDomain**, which is identifier to locate user in UG LDAP.

If **attemptLogin()** method is successful, **login()** method sets **userID** and **userDomain** info back into HTTP request by calling callback object's **setUserInfo()** method:

```
mcb.setUserInfo(userInfo[0], userInfo[2]);
```

Here, **userInfo[0]** is unique identified to locate user in UG LDAP. For example, uid and userInfo[2] is domain/organization name in UG LDAP under which user entry is available. This method sets this information as parameters in the HTTP request attribute:

```
public void setUserInfo(String uid,String domain){
        req.setAttribute("loginID", uid);
        req.setAttribute("userDomain", domain);
    }
```

The authentication framework uses the **loginID** and **userDomain** to get the information from the request. All custom modules must use same names for these parameters. This is mandatory for Convergence' authentication framework. The login() method returns true. Now JAAS framework will call **commit()** method of **LoginModule**, where **UserPrincipal** object is populated into authenticated **Subject**

object. This is mandatory for Convergence' authentication framework.

```
UserPrincipal userPrincipal = new UserPrincipal(mcb.getUserName());

    if (!subject.getPrincipals().contains(userPrincipal)) {
         subject.getPrincipals().add(userPrincipal);
    }
```

Here, **UserPrincipal** object takes **userName** of the user, which is nothing but unique identifier used to locate user entry in UG LDAP.

On successful completion of the commit() method, the control goes back to Convervgence' authentication framework. This step marks the end of the authentication process. The authentication framework now has all the required information like: **loginID**, **userDomain** and authenticated **Subject** with **UserPrincipal** objects. The Convergence authentication framework then loads the user from the UG LDAP.

Compiling the Sample Custom Module

If you must change the core class file names provided in this section, you must appropriately refactor the code. Some files use objects created by other core classes of the custom authentication module.

> **Note:** The paths used in the following example may differ for your installation.

1. In a temporary directory in **/com/sun/comms**. For example, create **/com/sun/comms/test**.

   > **Note:** The JAR file must be created by following the Java packaging layout rules. For example, the classes in this sample are packaged as **com.sun.comms**. So the Java files must be copied under the directory structure: **com/sun/comms**.

2. Copy the sample code provided earlier into the files **AppTestCallbackHandler.java**,**SunTestAuthCallBack.java**, and **SunTestLoginModule.java** under **/com/sun/comms/ test** directory.

3. Compile the java class files.

   ```
   cd /temporary_directory/com/sun/comms/test
   javac -classpath /opt/sun/comms/iwc/web-src/server/WEB-INF/lib/iwc.jar:/opt/
   sun/comms/iwc/web-src/server/WEB-INF/lib/
   commons-logging-api-1.1.1.jar:Glassfish_Home/glassfish/lib/javaee.jar
   AppTestCallbackHandler.java SunTestAuthCallBack.java SunTestLoginModule.java
   ```

4. Create a JAR archive.

   ```
   cd temporary_directory
   jar -cvf SunTestLogin.jar com
   ```

   > **Note:** If your custom authentication module requires any additional JAR files or classes, these must be bundled along with the JAR file.

5. Place the JAR file in the following directory:

   Convergence_Domain/applications/iwc/WEB-INF/lib

   Note: The custom authentication module must be on the system that can be accessed by Oracle GlassFish Server. It is best to place the JAR file on a location outside of the Convergence installation or deployed directories. See your Oracle GlassFish Server documentation for more information.

Configuring the Sample Custom Authentication Module

This section describes the steps to configure the custom authentication module with Convergence. Since this example comes bundled with Convergence server, all that is needed is to configure Convergence by setting the appropriate configuration parameters. The following are the instructions to enable the custom authentication module to use a file based authentication store.

1. Set the **auth.custom.service** name parameter in Convergence to indicate that a custom authentication module is being used.

   ```
   iwcadmin -o auth.custom.servicename -v "JAAS_CUSTOM"
   ```

2. Set the **auth.custom.loginimpl** parameter to the login module implementation created for custom authentication module.

   ```
   iwcadmin -o auth.custom.loginimpl -v "com.sun.comms.test.SunTestLoginModule"
   ```

3. Set the **auth.custom.callbackhandler** parameter to the custom callback handler used for the custom authentication module.

   ```
   iwcadmin -o auth.custom.callbackhandler -v
   "com.sun.comms.test.AppTestCallbackHandler"
   ```

4. Set the **auth.misc.CredentialFile** parameter to the directory where the authentication store is available. In this case, the authentication store is a file.

   Note: Here, the value of **auth.misc.CredentialFile** is case sensitive. While reading these parameters inside custom authentication module the name should match the configuration.

```
iwcadmin -o auth.misc.CredentialFile -v "/var/opt/SUNWiwc/config/userinfo.txt"
```

If you have created a custom authentication module for a different authentication store, you must follow the steps described below to enable the authentication module to work with Convergence.

1. Compile the custom authentication module source files and bundle them as a Java archive. See "Compiling the Sample Custom Module".
2. Configure Convergence to use the custom authentication module.
   a. Set the **auth.custom.service** configuration parameter to "JAAS_CUSTOM".
   b. Set the **auth.custom.loginimpl** configuration parameter to the custom login module implementation of the authentication module
   c. Set the **auth.custom.callbackhandler** to the call back handler of the custom authentication module.

d. Set any miscellaneous parameters that you have used for your custom authentication module by setting the **auth.misc** configuration parameter.

3. Deploy the custom module. See "Deploying the Authentication Module in GlassFish Server".

Deploying the Authentication Module in GlassFish Server

Restart the GlassFish server so that the module is updated in GlassFish Server's classpath.

Debugging and Troubleshooting the Custom Authentication Module

This section provides instructions on how to debug and troubleshoot the authentication module.

1. Set Convergence logging to **DEBUG** level.

```
iwcadmin -o log.AUTH.level -v DEBUG
```

2. Restart the GlassFish server.
3. Use the **tail** command to see the log messages generated.

Disabling the Custom Authentication Module

To change the custom authentication module to the default authentication module (LDAP) run following command.

```
iwcadmin -o auth.ldap.enable -v true
```

Restart the GlassFish server to ensure that the changes take effect in your deployment.

Summary

This section provides a recap of how to create a custom authentication module.

- Every custom authentication module should implement the following three classes:
  - A class that implements LoginModule interface
  - A class that extends HttpCallBackHander class
  - A (set of) class(es) that implements CallBack interface

  If your custom authentication module requires other classes that are specific to your implementation of the authentication module, the classes must be implemented.
- The **iwc.jar** should be there in classpath, while developing custom authentication module as it uses few Convergence specific classes like HttpCallBackHandler and UserPrincipal.
- As a best practice, it is good to bundle all dependent classes in a jar file. These should be made available in the web container's class path.
- Implementation of LoginModule interface and HttpCallBackHander class needs to be configured using the command line interface.
- Any additional configuration specific to custom authentication module can be configured as Misc parameter using CLI

- The custom authentication module must set two HTTP request attributes, userid and userDomain after successful authentication.
- The userDomain must be a valid domain entry in UG LDAP under which, Convergence can uniquely locate user entry by using user id as an identifier.
- The custom authentication module must create UserPrincipal object using userid and set it in Subject after successful authentication.

A

Convergence Secure Deployment Checklist

The following security checklist provides guidelines to help you secure Oracle Communications Contacts Server and its components.

Secure Deployment Checklist

- Install only the components you require.
- Lock and expire default user accounts.
- Use a strong LDAP password policy for user authentication.
- Enable data dictionary protection on the Oracle Database for Contacts Server.
- Restrict, control, and revisit user privileges:
  - Grant only the necessary privileges to each user.
  - Revoke unnecessary privileges from the PUBLIC user group.
  - Restrict permissions on run-time facilities.
- Enforce the use of access controls by using the Authorization Policies.
- Require clients to authenticate.
- Restrict network access by doing the following:
  - Use firewalls.
  - Never leave an unnecessary hole in a firewall.
  - Password-protect the Oracle listener against remote access.
  - Monitor listener activity.
  - Monitor who accesses your systems.
  - Restrict system access by IP addresses.
  - Encrypt network traffic.
- Apply all security patches and workarounds.
- Encrypt sensitive information.
- Contact Oracle Security Products if you discover a vulnerability in any Oracle product.