Software AG Infrastructure Administrator's Guide

Innovation Release

Version 10.2

April 2018

This document applies to webMethods Product Suite Version 10.2 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: SAG-INFRA-102-20180606

Table of Contents

About this Guide

This guide explains how to administer the Software AG Infrastructure used by many products.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at "http://documentation.softwareag.com". The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at "https://empower.softwareag.com".

To submit feature/enhancement requests, get information about product availability, and download products, go to "Products".

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the "Knowledge Center".

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at "http://techcommunity.softwareag.com". You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

1 Concepts

Software AG Common Platform

The Software AG Common Platform is OSGi-based and offers the possibility to dynamically construct executable instances of various products. It enables applications to be remotely installed, started, stopped, updated, and uninstalled without the necessity of a reboot. Packages and classes can be managed in great detail.

Considerations When Installing the Common Platform on Windows

Starting with version 10.1, you can install the Common Platform under Program Files on Windows. However, such a Common Platform installation will have a limited functionality. The installation scenario is supported to enable Composite Application Framework (CAF) to use ceratin libraries from the Common Platform if you install CAF under Program Files.

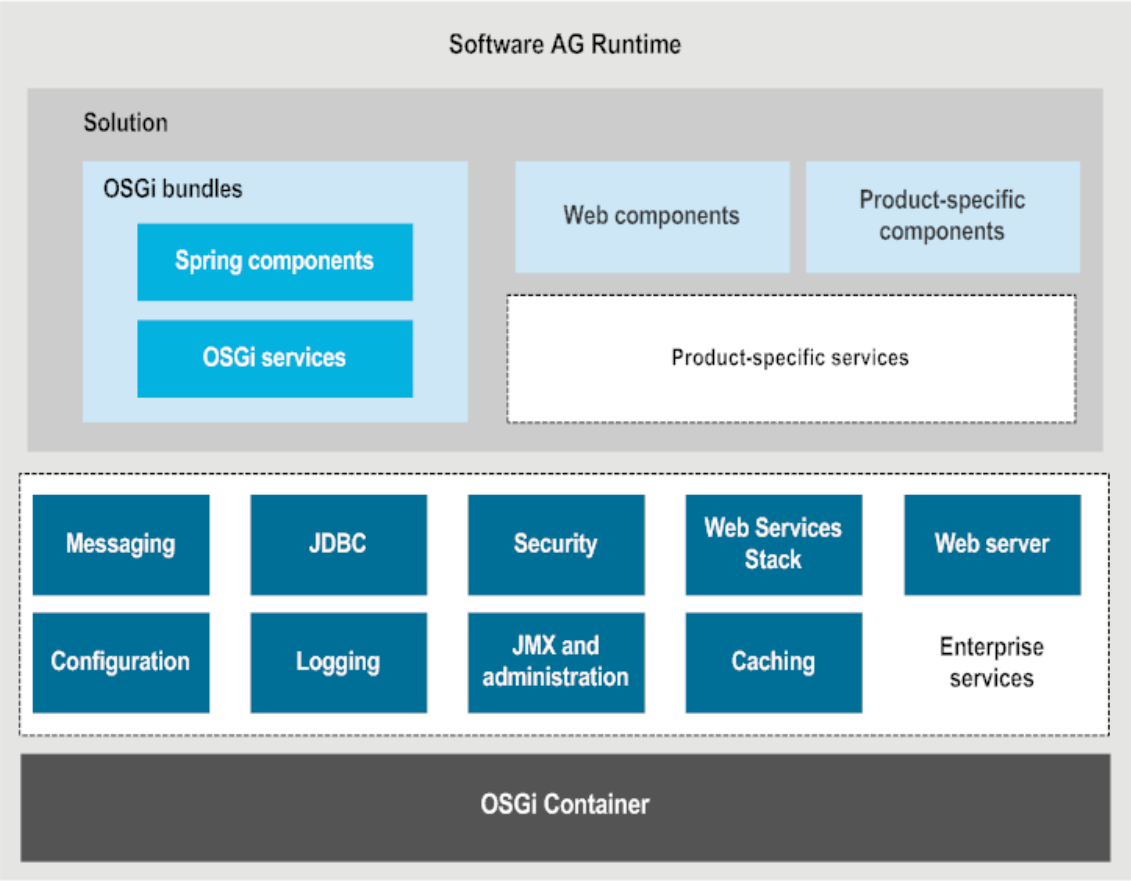
Software AG Runtime

Software AG Runtime is a runnable instance of the Common Platform environment. Software AG Runtime hosts and runs web applications. Software AG Runtime includes the following key components:

- Software AG Web Server based on Apache Tomcat.
- Authentication infrastructure named Software AG Security Infrastructure.
- Toolkit for creating, configuring, deploying, and managing web services named Software AG Web Services Stack.
- Event infrastructure named Software AG Digital Event Services. Enables Software AG products to communicate by exchanging digital events, which are typed and serialized data structures that convey or record information about the execution of a runtime. This information can be application information, such as the state of a business process step and associated business data, or system information, such as the amount of memory and number of threads an application is using.
- Spring Framework.

webMethods EntireX and CentraSite use Software AG Runtime. Terracotta BigMemory client libraries are integrated with Software AG Runtime and are therefore available to these products. Other Software AG products such as Command Central and Integration Server have their own instances of parts of Software AG Runtime in product-specific *profiles*.

You can use the OSGi technology supported by the Common Platform to construct your own applications from reusable components, and then run them within Software AG Runtime.



Software AGSecurity Infrastructure

Security Infrastructure provides security components for authentication of users, management of roles, and query of user, role, and group information. It works both on client-side applications and on server-side applications. Security Infrastructure is used by many Software AG products and can be used by your own applications.

Security Infrastructure's basic advantage is the re-use of existing security components. For example, Security Infrastructure supports the same security mechanism for an application that uses a database and another application that uses LDAP directory without any change of code on the application level.

Security Infrastructure is based on *login modules*, *login context*, and *JAAS configuration files*, which in turn are all based on the Oracle JAAS framework.

Login modules are reusable entities that define authentications to perform. Applications can call login modules to authenticate users; verify client certificates; or query user, role, or group information in user repositories. Security Infrastructure provides predefined login modules and OSGi services that you can configure for your environment and desired authentication process. You can also create your own login modules by copying predefined modules and modifying the copies.

You list login modules in login contexts. If you want an application to use more than one login module, you list multiple login modules in a login context.

You define login contexts in a JAAS configuration file. You set up one JAAS configuration file per JVM.

JAAS offers these benefits:

- Authentication is independent of applications.
- Professional services do not need special know-how to customize and re-use login modules for different authentication schemes.

JAAS accommodates the information for groups and roles in classes derived from java.security.Principal. The Principal interface represents the abstract notion of a Principal that can be any entity, such as an individual, a corporation, and a login ID, while the Subject class represents a grouping of related information for a single entity. Such information includes the Subject's identities, as well as its security-related attributes (passwords and cryptographic keys). If authentication is successful, JAAS creates a Subject that contains one or more Principals with security-related attributes like passwords and cryptographic keys. For example, if a Subject is a person named John, he may have two Principals:

- Principal 1 represents John as the citizen of a particular country.
- Principal 2 represents John as the employee of a particular company.

Both Principals refer to the same Subject even though they have different names.

The authentication process is as follows:

1. An application instantiates a login context.
2. The login context consults the application configuration (realm) in the JAAS configuration file to load all login modules for the application.
3. The application invokes the login context’s login method to authenticate the user.
4. The login method invokes all loaded login modules as specified in the login context.
5. Each login module tries to authenticate the Subject. If successful, login modules associate relevant Principals and credentials with a Subject object that represents the subject being authenticated. If unsuccessful, login modules throw an exception or the authenticate method returns false.
6. The login context returns the authentication status to the application.
7. If authentication is successful, the application retrieves the Subject from the login context. If not successful, no login occurs and the Subject is empty and does not contain any Principals.

For background information relating to Security Infrastructure, see Java™ Platform, Standard Edition 7 API Specification, Java™ SE 7 Security Documentation, JAAS Reference Guide, JAAS Tutorials, Introduction to JAAS and Java GSS-API Tutorials.

Software AG Web Services Stack

Software AG Web Services Stack is a toolkit for creating, configuring, deploying, and managing web services. It handles the complex process of processing request and response messages between web services within Software AG products.

You can specify individual configuration settings for your web services. This enables you to modify their behavior at runtime and facilitate the correct invocation of the functionality they expose. You can configure the web services by providing advanced design settings, such as web services addressing, security, and transactional behavior (for example, the service should only be executed on HTTPS with encryption, and the client can only execute the service between 2 and 5 p.m. on Thursdays).

You can deploy your web services on the default Web Services Stack servlet container and run them locally or you can deploy them on a fully functional application server and consume the functionality using a variety of Web service clients.

Web Services Stack supports these web services standards:

- HTTP and SMTP for basic network transport services
- XML (Extensible Markup Language) as data format
- UDDI for web service registries
- WSDL for service descriptions
- SOAP for XML messaging and RPC
- SOAP with Attachments (SwA)
- SOAP MTOM/XOP
- WS-Policy and WS-Policy Attachment Specifications
- WS-RM Policy
- WS-Security Policy
- WS-MeX
- WS-Addressing
- WS-ReliableMessaging
- XML Schema
- XML Core (XML Language, DTD, DOM, XML Name Space)

Considerations When Installing Web Services Stack on Windows

Starting with version 10.2, you can install Web Services Stack under Program Files on Windows. However, such a Web Services Stack installation will have a limited functionality. The installation scenario is supported to enable Software AG Designer

to use ceratin libraries from Web Services Stack if you install Designer under Program Files.

The Java Service Wrapper

The Java Service Wrapper is an application developed by Tanuki Software, Ltd. Some Software AG products use the Java Service Wrapper to:

- Start and stop the Java Virtual Machines (JVM) in which they run. You can configure Java startup parameters such as heap size and classpath.
- Record the console output from the JVM in a log file. This log includes stack traces that the JVM produces when a process throws an exception and any thread dumps you generate from the JVM. The wrapper log is particularly useful when a webMethods product runs as a Windows service, because console output is not normally available to you in this mode. The log file is named wrapper.log.
- Monitor the JVM for various fault conditions and take a specified action when a fault occurs. You can do the following:
  - Detect a nonoperational (hung) JVM. After the Java Service Wrapper starts the JVM, it pings the JVM periodically to check whether it is operational. If the JVM does not respond to a ping within a specified interval, the Java Service Wrapper assumes that the JVM has stopped functioning and restarts it. Each Software AG product configures this feature differently; some disable it entirely.
  - Detect thread deadlocks in the JVM. A thread deadlock occurs when two or more threads try to lock resources in a manner that causes all threads to wait indefinitely. The Java Service Wrapper can monitor the JVM for a deadlock condition and take a specified action (for example, restarting the JVM) when the condition occurs. For most Software AG products, this feature is disabled by default.
  - Detect specified messages in the console output. The Java Service Wrapper can monitor the console output and take a specified action when a given string of text appears. This feature is often used to watch for out-of-memory messages.
- Enable you to generate a thread dump when the JVM is running as a service under Windows.

This guide discusses the Java Service Wrapper as it is used by Software AG products that run on the Software AG Common Platform. The documentation for a product might contain additional instructions for using the Java Service Wrapper for that product.

Note: For information about Software AG products that use the Java Service Wrapper but do not run on the Software AG Common Platform, see the documentation for those products.

2 Running Web Applications

Configuring Software AG Runtime Credentials

1. Change the default Software AG Web Services Stack credentials (see "Managing Web Services" on page 95).
2. Change the default credentials of the internal user repository (see "Creating or Editing Internal User Repository Files" on page 38).
3. Generate a Java keystore file with a key pair and certificate for the Tomcat HTTPS connector (see "About Configuring HTTPS Connectors" on page 19).

Changing the Default Software AG Runtime Keystore and Truststore

Change the default Software AG Runtime keystore.jks and platform_truststore.jks files to a custom key pair and corresponding certificate.

Note: If other users have access to this certificate, they might have managing access for Software AG Runtime through JMX.

1. Go to the *Software AG_directory*\common\conf directory.
2. You cannot create a keystore with an existing alias (default alias is ssos). Back up the default keystore.jks and platform_truststore.jks files to another directory, and then delete the files from the conf directory.
3. In the *Software AG_directory*\common\conf directory, open a command window. Create the keystore by running this command:

```
Software AG_directory
\jvm\jvm\bin\keytool -genkeypair -alias keystore_alias
-keystore keystore_path -storepass keystore_password -validity days_count
-keypass keystore_password -keyalg key_algorithm -keysize key_size
-sigalg signing_algorithm -storetype JKS
```

   The keytool prompts for information such as your name, company, and address.

   Note: Due to limitation of the Software AG single sign-on system (SSOS), the -storepass and -keypass values must be identical.

4. Show the details of the keystore you created on the command window by running this command:

```
Software AG_directory
\jvm\jvm\bin\keytool -list -v -keystore keystore_path
-storepass keystore_password
```

   Note the certificate information.

5. Export the certificate from the keystore you created by running this command:

```
Software AG_directory
```

```
\jvm\jvm\bin\keytool -exportcert -alias keystore_alias
-file certificate_path -keystore keystore_path -storepass keystore_password
-storetype JKS
```

6. Create a truststore by running this command:

```
Software AG_directory\jvm\jvm\bin\keytool -import -file certificate_path
-alias truststore_alias -keystore truststore_path
```

7. Show the details of the truststore you created on the command window by running this command:

```
Software AG_directory
\jvm\jvm\bin\keytool -list -v -keystore truststore_path
```

   Make sure the certificate information is identical to the certificate you noted earlier in this procedure. If it is not identical, remove the keystore and truststore and start again at step 3.

8. Update your SSOS configuration. Go to the *Software AG_directory*\profiles\CTP \configuration\com.softwareag.platform.config.propsloader directory, open the com.softwareag.sso.pid.properties file, and update your SSOS configuration.

Variables	Description	Default Value
certificate_path	Path for generated certificate.	*Software AG_directory*/ common/conf/default.cer
days_count	Integer value of days count of the certificate validity.	10957
key_algorithm	Algorithm for encryption of the keystore.	RSA
key_size	Keysize of the keystore keys.	2048
keystore_alias	Alias for the new keystore.	ssos
keystore_path	Path to the new keystore.	*Software AG_directory*/ common/conf/keystore.jks
keystore_password	Password for the new keystore.	manage
signing_algorithm	Algorithm for the certificate signature.	SHA512with RSA

Variables	Description	Default Value
truststore_alias	Alias for the new truststore.	ssos
truststore_path	Path to the new truststore.	*Software AG_directory*/common/conf/platform_truststore.jks
your_C	CountryName	DE
your_CN	CommonName	ssos
You_L	Locality	Unknown
Your_O	Organization	sag
your_OU	OrganizationalUnit	default
your_ST	StateOrProvinceName	Unknown

9. Go to the *Software AG_directory*/profiles/CTP/configuration/com.softwareag.platform.config.propsloader directory. Open the com.softwareag.sso.pid.properties file and edit these properties:

```
com.softwareag.security.idp.truststore.location
com.softwareag.security.idp.truststore.keyalias
@secure.com.softwareag.security.idp.truststore.password
```

The default truststore location is @path\:sag.install.area/common/conf/platform_truststore.jks, and the default alias and password are ssos and manage.

About Configuring HTTP Connectors

Software AG Runtime comes with a predefined HTTP connector. You can modify the predefined connector as described in the steps below. Do not delete the predefined connector.

You can also create HTTP connectors to use in addition to the predefined connector. You can create connectors in Software AG Command Central (see the *Software AG Command Central Help*) or you can copy the predefined connector and modify it a described in the steps below.

HTTP connectors support HTTP/1.1 protocol connections on a configured port. You can configure one or more connectors on different ports, and all web applications deployed on Software AG Runtime will be accessible through these port addresses. For more

information on HTTP connector configuration, see the Apache Tomcat 7.x configuration guide.

Modifying the Predefined HTTP Connector or Creating an HTTP Connector

1. Go to the *Software AG_directory*\profiles\CTP\configuration\com.softwareag.platform.config.propsloader directory.
2. Do one of the following:
   - To modify the predefined HTTP connector, open the com.softwareag.catalina.connector.http.pid-*port_number* .properties file in a text editor.
   - To create an HTTP connector, copy the predefined HTTP connector properties file and open the copy in a text editor.
3. You can modify these properties:

Property	Description
port	TCP port number on which the connector will create a server socket and await incoming connections. The port number must be unique among all connectors.
alias	Identifies the connector to Software AG Command Central. The alias must be unique across all HTTP connectors. The alias defaultHttp is assigned to the predefined connector and designates it as the primary HTTP connector.
enabled	Whether to enable or disable the connector. Valid values are true and false. The default is false (disabled).

4. You can also modify the other properties in the file. For more information about the available connector properties, see the Apache Tomcat documentation.
5. Save the file.
6. Rename the properties file by modifying the *port_number* in the file name to match the value you specified on the `port` field.

About Configuring HTTPS Connectors

Software AG Runtime comes with a predefined HTTPS connector. You can modify the predefined connector as described in the steps below. Do not delete the predefined connector.

You can also create HTTPS connectors to use in addition to the predefined connector. You can create connectors in Software AG Command Central (see the *Software AG Command Central Help*) or you can copy the predefined connector and modify it a described in the steps below.

HTTPS connectors support SSL/TLS-secured HTTP/1.1 protocol connections on a configured port. You can configure one or more connectors on different ports, and all web applications deployed on Software AG Runtime will be accessible through these port addresses. For more information on HTTPS connector configuration, see the Apache Tomcat 7.x configuration guide.

Modifying the Predefined HTTPS Connector or Creating an HTTPS Connector

1. Make sure you have a server certificate. You must set the Common Name (CN) of the certificate to the URL of the server, but without the `https://`. For example, for a server at `https://MyWebServer:8443/`, the CN is `MyWebServer`.
2. Go to the *Software AG_directory*\profiles\CTP\configuration\com.softwareag.platform.config.propsloader directory.
3. Do one of the following:
   - To modify the predefined HTTPS connector, open the com.softwareag.catalina.connector.https.pid-*port_number* .properties file in a text editor.
   - To create an HTTPS connector, copy the predefined HTTPS connector properties file and open the copy in a text editor.
4. You can modify these properties:

Property	Description
keystore	Valid keystore file. By default, the keystoreFile property points to the localhost_dont_use_in_production.jks keystore, located in the *Software AG_directory*\profiles\CTP\configuration\tomcat\conf. It is only a sample and must not be used for production purposes.
keystorePass	Password for the keystore.
keystoreType	Java keystore type. Software AG Runtime supports the JKS (default), PKCS1, and PKCS12 Java keystores.

Property	Description
port	TCP port number on which the connector will create a server socket and await incoming connections. The port number must be unique among all connectors.
alias	Identifies the connector to Software AG Command Central. The alias must be unique across all HTTPS connectors. The alias defaultHttps is assigned to the predefined connector and designates it as the primary HTTPS connector.
enabled	Whether to enable or disable the connector. Valid values are true and false. The default is false (disabled).

5. Save the file.
6. Rename the properties file by modifying the *port_number* in the file name to match the value you specified on the `port` field.
7. Reopen the properties file and do one of the following:
   - If you modified an existing connector, the `keystorePass` password is already secured. Change the value of the `keystorePass` property by replacing the secure token handle with a new plaintext password that will be secured in turn and will overwrite the previous password in the secure storage.
   - If you created a new connector, secure the `keystorePass`, `keyPass`, and `truststorePass` properties by adding @secure. prefix to the property key. For example, for `keystorePass`, add the prefix @secure.keystorePass=change_this_password. The next time the properties file configuration is loaded, Software AG Runtime will move the value of the keystorePass property to an encrypted secure storage on the file system under the*Software AG_directory*\profiles\CTP\configuration\security\passman directory and the configuration will be written back, replacing the value with a secure token that contains a handle from the secure storage instead of the original plaintext value.
8. Save the file.

Accepting an HTTPS Connection on the Client Side

To accept an HTTPS connection on the client side, you can do either of the following:

- Import the server certificate into your Internet browser truststore. In case of a PKI, import the CA certificate that issued the server certificate. If you are accessing resources through a Web server's HTTPS protocol from a Java client using Oracle JSSE, you must also set a truststore via the *-Djavax.net.ssl.trustStore* property and a truststore password via the *-Djavax.net.ssl.trustStorePassword* property. For example:

```
-Djavax.net.ssl.trustStore=<your_truststore_here>
-Djavax.net.ssl.trustStorePassword=<your_truststore_password_here>
```

- When you open an HTTPS connection in your Internet browser, you will be asked whether you trust the certificate. Click **Yes**.

About the Predefined JMX Connector

Software AG Runtime is installed with a predefined JMX connector that is used by Software AG Command Central to manage Software AG Runtime.

The connector is defined in the com.softwareag.jmx.connector.pid-*port_number* .properties file in the *Software AG_directory*\profiles\CTP\configuration \com.softwareag.platform.config.propsloader directory.

Do not edit the com.softwareag.jmx.connector.pid-*port_number* .properties file unless Software AG Global Support asks you to do so.

Creating a JMX Connector

In addition to using the predefined JMX connector, you can also create a JMX connector to monitor Software AG Runtime. You can copy the predefined JMX connector and modify it as described in the following procedure.

To create a JMX connector

1. Go to the *Software AG_directory* \profiles\CTP\configuration \com.softwareag.platform.config.propsloader directory.
2. Copy the com.softwareag.jmx.connector.pid-*port_number* .properties file and open the copy in a text editor.
3. Modify any of the following properties, as required:

Property	Description
`host`	The name of the local interface on which the connector listens. If you do not specify a host, the connector is open on all local interfaces.
`port`	Required. The port number on which the connector creates a server socket and waits for incoming connections. The port number must be unique among all connectors.
`jaasRealm`	Required. The JAAS realm to use for login.

Property	Description
`secure`	Whether to use SSL for the Remote Method Invocation (RMI) connection. Valid values are `true` or `false`. The default value is `false`.
`enabled`	Whether to enable or disable the connector. Valid values are `true` or `false`. The default value is `true`.
`permission`	The name of the permission that the authenticated user must have to establish a JMX connection. If you specify`*`, no permission check is executed.
`truststoreFile`	The absolute path to a valid truststore file. You specify a value for `truststoreFile` only if you set `secure` to `true`.
`truststoreType`	The Java truststore type. Software AG Runtime supports the JKS (default) and PKCS12 truststores. You specify a value for `truststoreType` only if you set `secure` to `true` and specified a value for `truststoreFile`.
`truststorePass`	The password for the truststore. You specify a value for `truststorePass` only if you set `secure` to `true` and specified a value for `truststoreFile`.
`keystoreFile`	The absolute path to a valid keystore file. You specify a value for `keystoreFile` only if you set `secure` to `true`.
`keystoreType`	The Java keystore type. Software AG Runtime supports the JKS (default) and PKCS12 keystores. You specify a value for `keystoreType` only if you set `secure` to `true` and specified a value for `keystoreFile`.
`keystorePass`	The password for the keystore. You specify a value for `keystorePass` only if you set `secure` to `true` and specified a value for `keystoreFile`.
`internalPort`	The port number that the connector uses for internal calls by RMI. The default value is the value that you specified for the `port` property. However, if SSL is

Property	Description
	enabled, you must specify a different value from the value of `port`.

4. Save the file.
5. Modify the *port_number* in the properties file name to match the value that you specified for the `port` property.
6. Reopen the properties file.
7. Add a `@secure.` prefix to the keys of the `keystorePass` and `truststorePass` properties.

   For example, for `keystorePass`, specify `@secure.keystorePass=change_this_password`.

   The next time the properties file configuration is loaded, Software AG Runtime moves the value of the `keystorePass` property to an encrypted secure storage on the file system under the *Software AG_directory*\profiles\CTP\configuration\security\passman directory and the configuration is written back, replacing the value with a secure token that contains a handle from the secure storage instead of the original plaintext value.
8. Save the file.

About Configuring JNDI Resources

The standard way for web applications to access resources from the external environment is to look up objects via JNDI. Software AG Runtime provides a JNDI injection framework that allows web applications to access dynamic Common Platform resources in a transparent way. The JNDI injection framework supports the standard elements resource-ref, resource-env-ref, and env-entry for resource definition. The resource is accessed from the Java code in the standard way. It is bound under java:comp/env namespace.

You can configure custom web applications to use JNDI resources in the standard way (that is, by declaring a resource reference in the WEB-INF/web.xml file that is contained in the web application war).

Configuring the JNDI Injection Framework

The JNDI injection framework in Software AG Runtime is configured and enabled by default. The configuration is stored in the Tomcat configuration files context.xml and server.xml. The files are located in the *Software AG_directory*\profiles\CTP\configuration\tomcat\conf directory.

The context.xml file defines a context listener of type com.softwareag.platform.catalina.jndi.ResourceInjector that has several parameters with default values. You can change these values.

Parameter	Description
applicationStartup Timeout	Required. Period, in milliseconds, that the injector will wait for the host bundle to become active. After the period expires, the injector will try to obtain the host BundleContext. If the context is not available, the injector will fail the application startup. The default is 300000.
applicationStartup Poll	Required. How often, in milliseconds, the injector will poll the state of the host bundle. The default is 1000.
injectionStartup Timeout	Required. Period, in milliseconds, that the injector will wait for all unbound resources to be injected. If this period expires and resources are missing, the injector will fail the application startup. The default is 30000.
serviceProxy Timeout	Required. Damping period, in milliseconds, of the service proxies. If a service tracked by a proxy is not available, the injector will block the caller thread for the specified number of milliseconds. The default is 10000.

The server.xml file defines how and when Software AG Runtime is to deploy web applications. You can change these values.

Parameter	Description
autoDeploy	Whether to automatically deploy web applications. The default is true.
deployOnStartup	Whether to deploy web applications during Software AG Runtime startup. The default is false.

Configuring JNDI Resources

Define JNDI resources using property files whose names start with com.softwareag.catalina.resource.pid (for example, com.softwareag.catalina.resource.pid-petstore.properties). Store the configuration files in the *Software AG_directory*\profiles\CTP\configuration \com.softwareag.platform.config.propsloader directory. You can use the properties listed below in the JNDI resources configuration.

Property	Description
context	Optional. Name of the web context of the application into which to inject the resource configuration (for example, `petstore`). If the property is missing, the resource configuration will be injected into all web applications.
factory	Required. Fully qualified name of the ObjectFactory to use to produce the resource object (for example, org.apache.tomcat.jdbc.pool.DataSourceFactory). To enable OSGi service injection, this property is set to `service`. You can set these properties as well: ■ filter: Standard OSGi LDAP service filter. For example, you could select a DataSource service using the filter (&(dbName=JPetStore)(dbType=Derby)). ■ timeout: Damping period, in milliseconds, for all proxies produced by this ObjectFactory. This property overrides the serviceProxyTimeout property of the ResourceInjector as specified in the global context.xml.
name	Required. Name under which to bind the resource in the java:/comp/env namespace of the web application. The value is relative. For example, jdbc/JPetStoreDB means the absolute name of the resource will be java:/comp/env/jdbc/JPetStoreDB.
type	Required. Fully qualified name of the resource class (for example, javax.sql.DataSource).
enabled	Optional. Indicates whether to have the JNDI injector process the resource configuration. Valid values are true (default) and false.
multiple address properties	Optional. Actual JNDI resource configuration; these are names of factory fields for which getters and setters are available. The number, name, and type of these properties depends on the concrete resource and ObjectFactory that is being defined. For additional information, see the Tomcat JDBC pool documentation.

The sample JNDI resource configuration below defines a DataSource to inject into the configured context.

```
com.softwareag.catalina.resource.pid-petstore.properties
     # JND    I injection configuration
     context=/petstore
     name=jdbc/JPetStoreDB
     type=javax.sql.DataSource
```

```
factory=org.apache.tomcat.jdbc.pool.DataSourceFactory
# Resource definition
maxActive=100
maxIdle=30
maxWait=10000
username=user
password=pass
driverClassName=com.softwareag.platform.jdbc.dd.SQLServerDriver
url=jdbc:wm:sqlserver://hostname:1433;databaseName=dbName
```

Configuring Environment Entries

Define environment entries using a dynamic configuration subsystem, typically property files whose names start with com.softwareag.catalina.env.pid (for example, com.softwareag.catalina.env.pid-petstore.properties). Store the configuration files in the *Software AG_directory*\profiles\CTP\configuration \com.softwareag.platform.config.propsloader directory. You can use the properties listed below in the environment entry configuration.

Property	Description
context	Optional. Name of the web context of the application into which to inject the resource configuration (for example, `petstore`). If the property is missing, the resource configuration will be injected into all web applications.
enabled	Optional. Indicates whether to have the JNDI injector process the resource configuration. Valid values are true (default) and false.
(1-9).name	Required. Name under which to bind the resource in the java:/ comp/env namespace of the web application. The value is relative. For example, jdbc/JPetStoreDB means the absolute name of the resource will be java:/comp/env/jdbc/JPetStoreDB.
(1-9).type	Required. Fully qualified name of the environment entry class (for example, javax.lang.String).
(1-9).value	Required. Value to return when this environment entry is looked up through JNDI by its name or injected as a @Resource.
(1-9).override	Optional. Indicates whether an environment entry in the web.xml can override the same environment entry defined in a more global configuration (for example, the context.xml or server.xml file). Valid values are true (default) and false.

Below is a sample environment entry configuration.

```
context=/petstore1.name=env/
JPetStoreEnvConfiguration1.type=java.lang.
```

```
String1.value=EnvConfigurationValue
```

Configuring the Software AG Runtime Java Service Wrapper

Software AG Runtime runs on the Software AG Common Platform, which in turn runs in a JVM. The JVM is launched by the Software AG RuntimeJava Service Wrapper.

See "Configuring the Java Service Wrapper" on page 97 for general information about the Tanuki Software, Ltd.Java Service Wrapper. Do not make any changes to the wrapper.conf file. Follow the instructions in "Editing Java Service Wrapper Properties" on page 98 to configure the Software AG RuntimeJava Service Wrapper. However, do not make any changes to the Software AG Runtime custom_wrapper.conf file other than the ones described below. The wrapper.conf and custom_wrapper.conf files are located in the *Software AG_directory*/profiles/CTP/configuration directory.

You can change the wrapper.java.initmemory and wrapper.java.maxmemory properties. The defaults for these properties are 256 and 512, respectively. If you set these properties to a non-zero value, the Java Service Wrapper adds an appropriate –Xms parameter. If you want to use the default values that are configured in the JVM itself, set these properties to 0 in the custom_wrapper.conf file. You can then set the –Xms parameter manually as an additional property in the custom_wrapper.conf file.

The JVM timeout, deadlock detection, and console filtering fault monitoring features are not enabled for Software AG Runtime. Do not enable them. Only modify the JVM timeout properties if asked to do so by Software AG for troubleshooting purposes.

Configuring Software AG Runtime Log Settings

Software AG Runtime delivers Journal Logging for logging purposes. To enable users to configure log settings, Software AG Runtime installation contains a log_config.xml, located in the *Software AG_directory*\profiles\CTP\configuration\logging directory.

Hot Configuration Update

Software AG Runtime runs a watchdog service that monitors the files under the *Software AG_directory*\profiles\CTP\configuration \com.softwareag.platform.config.propsloader directory and the JAAS configuration file at *Software AG_directory*\profiles\CTP\configuration \jaas.config and updates the runtime if changes to those files are detected. The watchdog maintains a set of configuration loaders for each supported file type (currently .properties and jaas.config). The poll interval for each configuration loader can be configured by modifying the *Software AG_directory*\profiles \CTP\configuration\com.softwareag.platform.config.propsloader \com.softwareag.config.watchdog.pid.properties file. The following properties can be configured:

Property	Description
poll.default	The poll interval in milliseconds to be used for configuration loaders that do not explicitly specify a poll interval. The default value is 5000. When the value of this property is set to less than 1000, polling is disabled for all loaders without values and all loaders with values set to less than 1000.
poll.file. property.loader	The poll interval in milliseconds to be used for property file configuration loader. No default value. When the value of the property is set to less than 1000, the value is reset to the value of the poll.default property.
poll.jaas. file.loader	The poll interval in milliseconds for the JAAS configuration file loader. The default value is 1000. When the value of the property is set to less than 1000, the value is reset to the value of the poll.default property.

Using Path Tokens

Software AG Runtime supports the usage of path tokens in the properties files under the *Software AG_directory*\profiles\CTP\configuration\com.softwareag.platform.config.propsloader directory and in the JAAS configuration file located at *Software AG_directory*\profiles\CTP\configuration\jaas.config.At runtime the path tokens are detected and replaced with the respective absolute location. These replacements take place in memory only and the files on disk will always contain values with path tokens.

The following standard path tokens are supported:

Token	Resolves to
osgi.install.area	*Software AG_directory*/profiles/*profile* /
osgi.instance. area	*Software AG_directory*/profiles/*profile* /workspace/
osgi.configur ation.area	*Software AG_directory*/profiles/*profile* /configuration/
sag.install.area	*Software AG_directory*

To specify that a path token must be resolved to a regular path, add a @path: prefix to the token name. If the path token must be resolved to an URL, add a @url: prefix instead.

The following examples are valid for the jaas.config file:

For a property that contains the Software AG installation directory (C:/SoftwareAG) as an absolute path (for example, someProperty=C\:/SoftwareAG/common/conf/someFile.conf), the property value can be modified by replacing C\:/SoftwareAG with @path:sag.install.area. After the change, the property will look like this: `someProperty=@path:sag.install.area/common/conf/someFile.conf`.

If the property contains an URL instead of an absolute path (for example, someProperty=file\:\C\:/SoftwareAG/common/conf/someFile.conf), the property value can be modified replacing file\:C\:/SoftwareAG with @url:sag.install.area. After the change, the property will look like this: `someProperty=@url:sag.install.area/common/conf/someFile.conf`

Important: When working in a .properties file you should use the \ symbol to escape symbols that may potentially break the configuration, for example, *:* .

Starting and Stopping Software AG Runtime

Your Software AG Runtime installation directory contains startup scripts which enable you to start and stop the Software AG Runtime instance. Different scripts are available depending on your operating system.

Starting and Stopping Software AG Runtime on a Windows System

The Software AG Runtime service is Software AG Runtime *release* . It is registered to start automatically at system start. You can start, stop, and modify the service in **Control Panel** > **Administrative Tools** > **Services**.

You can modify the startup type of the service in the **Services** window. The startup type of the service can be set to **Automatic**, **Manual**, or **Disabled**. The recommended startup type is **Automatic**.

Starting and Stopping Software AG Runtime on a UNIX System

Before you start the daemon processes on UNIX, you need to set sufficient data user limits for the shell which starts the Software AG Runtime daemons. Having an insufficient data user limit might result in an OutOfMemoryError java exception at startup. For more information on setting data user limits, see the main page for *ulimit* or contact your system administrator.

The installation registers the daemons for Software AG Runtime in the UNIX init structure so that Software AG Runtime starts automatically when the system starts. The

scripts below are installed, where `number` refers to a number which gets incremented by 1 for each installation on the local machine.

System	Scripts
Linux, Solaris	■ /etc/init.d/sag*number* ctp*release_number* ■ /etc/rc*system_runlevel* .d/K20sag*number* ctp*release_number* ■ /etc/rc*system_runlevel* .d/S60sag*number* ctp*release_number*
AIX	■ /etc/sag*number* ctp*release_number* ■ Entry in /etc/inittab: sag*number* ctp*release_number* :*system_runlevel* :wait:/etc/sag*number* ctp*release_number* start > /dev/console 2>&1
HP-UX	■ /sbin/init.d/sag*number* ctp*release_number* ■ /sbin/rc*system_runlevel* .d/K20sag*number* ctp*release_number* ■ /sbin/rc*system_runlevel* .d/S60sag*number* ctp*release_number*

To temporarily deactivate a service, remove or rename these files manually. Native configuration tools like the Yast Run-Level-Editor on Linux do not work.

The path to the Software AG Runtime daemon is *Software AG_directory*/common/bin/wrapper-3.5.25 and the daemon can have several child processes.

To start Software AG Runtime manually, start the daemon *Software AG_directory*/profiles/CTP/bin/startup.sh.

To stop Software AG Runtime manually, stop the daemon *Software AG_directory*/profiles/CTP/bin/shutdown.sh.

Managing Software AG Runtime Security

The Software AG Runtime security is managed by the jaas.config file located in the *Software AG_directory*\profiles\CTP\configuration directory. This security configuration file contains application contexts for the different parts of Software AG Runtime authentication. You can use the default login modules in the file or you can add your own modules that enable the use of SSO. The default authentication mechanism checks the username and password against the local user repository handled by the InternalLoginModule. The local user repository is in the users.txt file located in the *Software AG_directory*/common/conf/ directory.

For more information about available authentication mechanisms, see "Setting Up Security" on page 33.

3 Setting Up Security

Setting Up the JAAS Configuration File

Set up one configuration file per JVM. A JAAS configuration file comprises the following:

- One or more login contexts.
- One or more login modules in each login context. Login modules are listed in the order they should be called by the application.
- Classification of login modules, defined using flags such as required, requisite, or optional.
- Parameters that specify the type of authentication to use, such as check_crl_status=true.
- Comments that provide useful information about the file contents.

Different types of Principals are derived from an available Subject. The Principals architecture in Security Infrastructure is based on an abstract class called AbstractSagPrincipal, and all other SAG Principals extend it. Security Infrastructure provides some implemented classes for common use cases; these classes are SagUserPrincipal, SagGroupPrincipal, SagRolePrincipal, LightWeightPrincipal. Security Infrastructure returns no or only one user Principal for the authenticated user. Many applications expect one and only one SagUserPrincipal as the result of a successful authentication. However, a different expected behavior cannot be excluded. Make sure you configure the login contexts accordingly.

Creating the JAAS Configuration File

Go to the *Software AG_directory*/profiles/*profile* /configuration directory. Open a text editor and create a file named jaas.config.

Note: Store the JAAS configuration file in the directory specified above because files in those directories are automatically migrated during product upgrades. If you store a JAAS configuration file in a different location, you will have to remember to migrate the file manually.

Defining a Login Context

In the jaas.config file, define a login context. For example:

```
SoftwareAGSampleLoginContext {
```

Use semi-colons (;) to separate login contexts from each other.

Defining the Login Modules

In the login context, list the full class names of the login modules in the order the modules should be called by the application. List one classification flag after each login module name. List any parameters after the classification flag, separating the parameters with a space or a new line. Use semi-colons (;) to separate login modules from each other.

The code sample below shows a login context that contains the predefined login modules X509CertificateLoginModule and InternalLoginModule.

```
SoftwareAGSampleLoginContext {
    com.softwareag.security.jaas.login.modules.X509CertificateLoginModule required
  check_crl_status=true crl_url="${com.softwareag.security.crl.url}"
    truststore_url="${com.softwareag.security.truststore.url}"
    truststore_password="${com.softwareag.security.truststore.password}"
    truststore_type=jks overwrite_username=false;
      // Internal repository login module (java based)
       com.softwareag.security.jaas.login.internal.InternalLoginModule requisite
        template_section="INTERNAL"
        logCallback="true"
        internalRepository="@path:sag.install.area/common/conf/users.txt"
        create_group_principal="true"
        groupRepositoryPath="@path:sag.install.area/common/conf/groups.txt";
      // Role repository login module
      com.softwareag.security.authz.store.jaas.login.RoleLoginModule optional
        storage_location="@path:sag.install.area/common/conf/roles.txt";
};
```

You can also use the domain parameter in a login module. This parameter enables a dynamic use of login modules. When a user logs in to an application with a domain and user name, login modules that use the domain parameter verify the domain and begin the authentication process for the user only if the domain corresponds to the one defined for the login module.

Classification flags you can use are listed below.

Classification	Means the authentication specified in the login module . . .
Requisite	Must succeed. If the authentication succeeds, the authentication process proceeds down the login module list defined in the login context. If it fails, control is returned to the product and authentication stops.
Required	Must succeed. If the authentication succeeds or fails, the authentication process proceeds down the login module list defined in the login context. For example, you might want to execute audit login module that logs user login attempts. However, the overall authentication succeeds only if all requisite and required login modules succeed.

Classification	Means the authentication specified in the login module . . .
Sufficient	Does not have to succeed. If the authentication succeeds, control is returned to the product and authentication stops. If the previous requisite and required login modules also succeeded, the overall authentication succeeds. If the authentication fails, the authentication proceeds down the login module list defined in the login context.
Optional	Does not have to succeed. If the authentication succeeds or fails, the authentication process proceeds down the login module list defined in the login context. If there are no requisite or required login modules in the login context, the overall authentication succeeds only if the authentication specified in at least one sufficient or optional login module succeeds.

The parameters below are global parameters that apply to all types of login modules. You can use them in all login modules developed using the SagAbstractLoginModule.

Parameter	Description
create_user_ principal	Optional. Used to define whether the commit () method creates a SagUserPrincipal using the SagCredentials available in the sharedState Map. Valid values are: true - The commit () method creates a SagUserPrincipal. If you set this parameter to true, it cannot later be changed. false - The commit () method does not create a SagUserPrincipal. The login modules that do not create SagUserPrincipal in their own commit () method must call the super.commit () method.The SagUserPrincipal is created only once. This is the default.
store_credentials	Optional. Used to define whether to store SagCredentials in Subject.privateCredentials. The servlet context and header field of SagCredentials are not stored. Valid values are: true - SagCredentials is stored in Subject.privateCredentials. This is the default. false - SagCredentials is not stored in Subject.privateCredentials. Keeping the password in clear text in the Subject.privateCredentials may constitute a security risk, depending on how the Subject is handled. However, there are use cases where the password needs to be accessible through the Subject. Store the password only if necessary.

Parameter	Description
keep_password	Optional. Used to define whether to keep the password (if present in SagCredentials) in the credentials that are stored in Subject.privateCredentials. Valid values are: true - if present in the SagCredentials, the value is kept in the credentials that are stored in the Subject.privateCredentials. The default value is true. false - if present in the SagCredentials, the password is not kept in the credentials that are stored in the Subject.privateCredentials. This parameter requires the store_credentials parameter to be set to true.

For a complete list of parameters you can use in login modules, see "Predefined Login Modules" on page 47. The domain parameter is listed in the predefined InternalLoginModule and LDAPLoginModule.

You can use location tokens (@path and @url) on parameters that call for paths or URLs. For more information about path token support, see "Running Web Applications" on page 15.

Verifying JAAS Configuration

Make sure all paths and URLs in the JAAS configuration file are valid. All paths and URLs use the PluggableUI LoginContext; make sure that login context is set up correctly.

Turning On Logging

Security Infrastructure uses the log4j package for logging data. To turn on logging, include these properties in the properties list of the first login module of the stack in the login context in the JAAS configuration file:

```
useLog="true"
logLevel="debug"
logFile="full_path_to_log_file"
```

The resulting file contains the entire debug information generated during the login process, role management, and user repository management.

You can configure Security Infrastructure login modules to log information into an external file on the file system. Make sure the directory is not write-protected for the user who executes the JVM. On UNIX-based operating systems, Software AG recommends using the /tmp directory.

Software AG recommends that you turn off the logging after you collect sufficient information about the issues. If you do not change these logging settings, the system

keeps logging information to the log file, which leads to greater file size and reduced overall performance. Alternatively, instead of configuring external logging on Security Infrastructure, you can also check the system logging.

Making the JAAS Configuration File Active

If you are using Security Infrastructure with Software AG Runtime, go to the *Software AG_directory*/profiles/CTP/configuration directory and open the config.ini file. Set the java.security.auth.login.config property to the URL for the JAAS configuration file. For example:

```
java.security.auth.login.config=@url\:osgi.configuration.area/jaas.config
```

If you are not using Security Infrastructure with Software AG Runtime, set the java.security.auth.login.config Java system property to the URL for the JAAS configuration file. The property can be set by the application at start up programmatically or as a parameter of a JVM. For example:

```
-Djava.security.auth.login.config=URL_for_jaas.config_file
```

Creating Technical User Credential Files

The Security Infrastructure JAAS stack provides the SagCredentials class. Security Infrastructure login modules support only this type of credentials. SagCredentials are queried by SagCallbackHandler, which is the default callback handler for credentials. It supports SagCredentialCallback. Upon successful authentication, the SagCredentials can be stored as private credentials in the Subject, from which they can be retrieved by the application. Following is a list of user's attributes that SagCredentials sets and retrieves.

- Domain name, password, and user name
- X.509 certificate chain including user certificate and the issuer certificate (excluding the root certificate)
- SAML artifact
- Netegrity SiteMinder token
- HTTP header fields

Creating or Editing Internal User Repository Files

You can create or edit internal user repository files that contain user names and encrypted passwords using the Security Infrastructure Internal User Repository Command Line Tool. Files created with the Internal User Repository Command Line tool can be used with the InternalLoginModule.

Open a command window and go to the *Software AG_directory*/common/bin directory. Start the tool using the appropriate command below.

System	Command
Windows	internaluserrepo.bat [-f *file*] [-c] [-d | -e] *user_Id* [-p *password*]
UNIX	./internaluserrepo.sh [-f *file*] [-c] [-d | -e] *user_Id* [-p *password*]

The arguments for the command are described below.

Argument	Description
-h	Print guidelines for using the tool.
-c	Create or edit a text repository file. To create a file named users.txt in the *Software AG_directory*/common/bin directory, specify -c but not -f. To create a file with a specific name and location, or to modify an existing file, specify - c and -f.
-f *file*	Location and name of the file to create or modify.
-d *user_Id*	Deletes the credentials for the specified user from the file.
-e *user_Id*	Change the password for the specified user ID.
user_Id	If you have a users.txt file in the *Software AG_directory*/common/bin directory, use this argument without -d or -e to add a new user to the file. User names can contain up to 128 digits, Latin letters, and the characters ! ( ) - . ? [ ] _ ~.
-p *password*	Password for the specified user ID. Passwords can contain up to 128 digits, Latin letters, and the characters ! ( ) - . ? [ ] _ ~. If you do not specify this argument, the tool will prompt for the password.

If the command fails, the tool returns the appropriate exit code.

Exit Code	Description
-1	User ID specified on -e argument not found in the repository file.
1	Password is not set. Specify a password.

Exit Code	Description
2	User ID is too long.
3	User ID contains an invalid character.
4	Password contains an invalid character.
5	Password is too long.
6	Repository file lists more than one version.
7	Repository file lists a version in an unknown format.
8	Repository file does not list any version.
9	User does not have permissions required to create of modify the repository file.
10	User ID not specified on the command.
11	Specified parameters conflict or are invalid.

Creating Login Modules

Security Infrastructure consists of a set of bundles located in the *Software AG_directory*/common/runtime/agent/repository/plugins directory. Security Infrastructure bundle names start with com.softwareag.security.sin. All interfaces and common classes are contained in com.softwareag.security.sin.common_*release_number* .jar.

You can create login modules by copying predefined modules and modifying the copies.

All LoginModules must extend the SagAbstractLoginModule. This class is an abstract superclass for all Security Infrastructure LoginModules. It handles the retrieval of credentials for all derived classes and the handling of the inter-LoginModule SSO. Derived classes have to implement initConfiguration () and authenticate (). See the Security Infrastructure Javadoc for details.

Important: When you extend the SagAbstractLoginModule, do not overwrite the initialized () method. If you need to overwrite it (for example, when you use a new Callback and CallbackHandler), explicitly invoke the super.initialize () method instead. This prevents the failure of other Security Infrastructure-based login modules.

To write a LoginModule using SagAbstractLoginModule, define the parameters for the new module. Extend SagAbstracLoginModule with main focus on the implementation of initConfiguration () and authenticate (). The first method gets the incoming parameters from the JAAS configuration file in the following way:

```
String optionValue = (String) options.get(OPTION_VALUE);
```

The second method takes care of the actual authentication of the user. It is called by the login () method from the SagAbstracLoginModule. You can modify the user credentials according to the inter-LoginModule SSO.

If you want to implement other methods from the SagAbstracLoginModule (for example, logout() or commit()), it is a good idea to invoke the super method from the parent class at the end.

Using the LDAP Framework

LDAP framework is an OSGi service that uses dynamic configuration properties files to configure an LDAP directory. The default dynamic configurations properties file is stored in the *Software AG_directory*\profiles*profile_name* \configuration \com.softwareag.platform.config.propsloader directory. The aliases from these files are used in the JAAS configuration file.

The LDAP configuration behavior depends on the URL property in the JAAS configuration file. The following behavior patterns exist:

Pattern	LDAP Behavior
URL property is set in jaas.config, but no aliases are set	LDAP login module uses only the server configured via the JAAS configuration file.
URL property is not set in jaas.config, and no aliases are set	LDAP login module uses all servers configured via the LDAP dynamic configuration.
URL property is not set in jaas.config, but aliases are set	LDAP login module uses only the servers configured via the LDAP dynamic configuration with matching aliases.

These properties are used with their default values the first time you start your product. The dynamic configuration properties files must follow specific naming conventions. The following table outlines the dynamic configuration properties for all LDAP connections.

Parameter	Description
watt.server.ldap. DNescapeChars	String. Specifies which characters to escape when building LDAP queries. Valid values: all symbols. No default.
watt.server.ldap. retryCount	Long. Specifies how much retries can be performed on LDAP connections before giving up. Valid values are any positive Long number. The default value is 0.
watt.server.ldap. DNstripQuotes	Boolean. Specifies whether to remove quotes when building LDAP queries. Valid values are true (default) or false.
watt.server.ldap. extendedProps	String. Specifies the additional JNDI properties to be set. No default.
watt.server.ldap. retryWait	Long. Specifies how many milliseconds to wait between retries. Valid values are any positive Long number. The default value is 0.
watt.server.ldap. doNotBind	Boolean. Specifies whether the login module should perform an actual binding to LDAP servers. Valid values are true or false (default).
watt.server.ldap. DNescapePairs	Pair of strings. Specifies whether to escape substitutions. Each time the login module meets the first member of the pair, it replaces it with the second member. Valid values are pairs. All string of characters are valid values for the members of the pair. No default.
watt.server.ldap. DNescapeURL	Boolean. Specifies whether to escape the URL when building LDAP queries. Valid values are true or false (default).
watt.server.ldap. ignore.server CertificateValidity	Boolean. Specifies whether the login module should ignore the error if it uses SSL but the server certificate is expired or not yet valid. Valid values are true or false (default).
watt.server.ldap. extendedMessages	Boolean. Specifies whether JNDI should use extended messages. Valid values are true or false (default).
watt.server.jndi. searchresult. maxlimit	Long. Specifies the maximal number of results the jndi can return when a search is performed. Valid values are any positive Long number. The default value is 0 (no limit).

Parameter	Description
watt.server.ldap.includeOnly ActiveGroups	Boolean. This option applies only to Integration Server. It is not used in the LDAP Framework. The login module uses this option to remove from the memory those groups that do not belong to both ACL and LDAP. Valid values are true (default) or false.

Updating the Single Sign-On System for Your Product

The Single Sign-On (SSO) service issues and parses a signed SAML assertion that can be used as a single sign-on and delegation token. The default implementation uses the SAML 2 assertion issuance, however SAML 1.1 version is supported as well.

The bundles required for the SSO service are available within all Common Platform profiles. The SSO service requires a dynamic configuration properties file in order to work correctly. By default, your installation contains a com.softwareag.sso.pid.properties file in the *Software AG_directory*/profiles/*profile_name* /configuration/com.softwareag.platform.config.propsloader directory.

The following table outlines the parameters of the SSO service dynamic configuration.

Parameter	Description
com.softwareag.security.idp.keystore.location	Location of the keystore to use. Default is /common/conf/keystore.jks.
com.softwareag.security.idp.keystore.password	Optional. Password for the keystore to use.
com.softwareag.security.idp.keystore.type	Optional. Type of the keystore. Valid values are PKCS7, PKCS12, or JKS (default).
com.softwareag.security.idp.keystore.keyalias	Key alias to use for signing. Default is ssos.
com.softwareag.security.idp.truststore.location	Optional. Truststore to use.
com.softwareag.security.idp.truststore.password	Required if com.softwareag.security.idp.truststore.location is specified. Truststore password.

Parameter	Description
com.softwareag.security.idp.truststore.type	Required if com.softwareag.security.idp.truststore.location is specified. Type of the trustore. Valid values are PKCS7, PKCS12, or JKS (default).
com.softwareag.security.idp.truststore.keyalias	Truststore key alias. Default is ssos.
com.softwareag.security.idp.assertion.lifeperiod	Time to live for the issued assertion (in milliseconds). Default is 300.
com.softwareag.security.idp.ehcache.location	Location in which to cache the configuration used for caching incoming SAML assertions.

Go to the *Software AG_directory*/profiles directory. In each *profile_name* /configuration/com-softwareag.platform.config.propsloader directory, open the com.softwareag.sso.pid.properties file and edit these properties:

```
com.softwareag.security.idp.truststore.location
com.softwareag.security.idp.truststore.keyalias
@secure.com.softwareag.security.idp.truststore.password
```

The default truststore location is @path\:sag.install.area/common/conf/platform_truststore.jks, and the default alias and password are ssos and manage.

If you are editing the Command Central profile (profile name CCE) or Platform Manager profile (profile name SPM), also edit these properties:

```
com.softwareag.security.idp.keystore.location
com.softwareag.security.idp.keystore.keyalias
@secure.com.softwareag.security.idp.keystore.password
```

The default keystore location is @path\:sag.install.area/common/conf/keystore.jks, and the default alias and password are ssos and manage.

Creating Custom Keys and Certificates

Software AG Common Platform provides a single sign-on service that has a predefined keystore (keystore.jks) and truststore (platform_truststore.jks). The predefined keystore and truststore contain default keys for issuing and validating signed SAML assertions. You can create and modify the keystore and certificates using the certtool tool provided by Security Infrastructure. The certtool is located in the *Software AG_directory*\common\bin directory and the file is named certool.{bat|sh} file. It is a wrapper of Java keytool and has default options that are used if you do not provide any custom input.

The options in the certool are mostly self-explanatory. The DEFAULT_PATH option indicates the default path in which the certificate stores are created when you install your products. The SIG-ALGORITHM option specifies the algorithm to use to sign the

self-signed certificate if you make any changes. The algorithm must be compatible with KEY_ALGORITHM. The value of SIG-ALGORITHM is derived from the algorithm of the underlying private key. For example, if the private key is of type DSA, the value of the SIG_ALGORITHM option is SHA1withDSA.

Important: The options have reasonable default values. If you modify them, use extreme caution; if incorrect values are entered, Security Infrastructure might stop working.

After you create a new certificate and add it to the keystore, you must update the configuration of the single sign-on service (SSOS) for your changes to take effect. If the keystore file already exists, and you try to generate a new key pair in the same keystore file, the certool warns that the file will be overwritten.

Open a command window and go to the *Software AG_directory* \common\bin directory. Start the certtool using the appropriate command below.

System	Command
Windows	`certtool.bat`
UNIX	`./certtool.sh`

Below are the arguments you can specify on the certtool command.

Argument	Description
-listkeystore	Lists keystore certificates currently located in the keystore. The default keystore certificate is keystore.jks with a default password of manage. The keystore should contain only one keystore certificate that is used for issuing signed SAML assertions.
-listtruststore	Lists truststore certificates currently located in the truststore. The default certificate is platform_truststore.jks with a default password of manage. The truststore can contain multiple public truststore certificates that are used for validating SAML assertion signatures.
-add	Adds a trusted certificate to the truststore. The .cer file is added to the location specified by the TRUSTSTORE_FILE option. If the truststore only contains the platform_truststore.jks certificate, then platform_truststore.jks is used.

Argument	Description
-delete	Deletes a trusted certificate from the truststore. You are prompted to provide the alias name of the certificate file to delete.
-generate	Generates a key pair and exports the public information as a .cer file. You are prompted to provide a common name (CN) for the certificate. The keystore certificate is generated in the location specified by the DEFAULT_PATH option. **Note:** The specified password will be used for both the keystore and the key.

Developing a JAAS Client

Create the login context. Below is an example of how to authenticate a user. In this case, you must instantiate a LoginContext, where *configuration_entry* is the name used as the index into the JAAS configuration file:

```
import javax.security.auth.login.LoginContext;. . .LoginContext
loginContext = new LoginContext(configuration_entry_name,
CallbackHandler_to_be_used_for_user_interaction);
```

Troubleshooting Problems

Verifying the JAAS Configuration

Make sure all paths and URLs in the JAAS configuration file are valid. All paths and URLs use the PluggableUI LoginContext; make sure that login context is set up correctly.

When Problems Persist

If you still have problems logging in, or can log in but do not have enough rights to use a certain product, install and run the Testjaas web application. Testjaas troubleshoots Security Infrastructure login modules.

1. Go to the Software AG Community Website > Suite Downloads at "http://techcommunity.softwareag.com/ecosystem/communities/public/webmethods/products/suite/downloads/" and download the testjaas.war.
2. Install the testjaas.war in the *Software AG_directory*\profiles\CTP\workspace\webapps directory.
3. Point a browser to http://*host* :*port* /testjaas/testjaas and save the output in a file. You can manually verify the working of your login context by pointing the browser to

http://*host* :*port* /testjaas/InputForm.html and by providing the login context and the logon credentials.

4. Save the output in a file and send the file to Software AG Global Support.

Predefined Login Modules

SagAbstractLoginModule

SagAbstractLoginModule is the basic login module in Security Infrastructure. It provides you with a commit() method that uses the global configuration parameters. See "Defining the Login Modules" on page 35 for details.

You can extend this login module to create your own login modules. You can use this login module to create the SagUserPrincipals with the information stored in the shared map through the authentication process.

When setting up the JAAS configuration, keep in mind the following basics:

- The Security Infrastructure-based login contexts return zero or only one SagUserPrincipal if the authentication succeeds. When setting up the JAAS configuration, keep in mind that some applications expect only one SagUserPrincipal as the result of a successful authentication. If your application expects more than one user principal, you must configure the login context accordingly.
- Keeping the password in clear text in the Subject.privateCredentials may constitute a security risk, depending on how the Subject is handled. However, there are use cases where the password needs to be accessible through the Subject, so you must store the password only if needed.

InternalLoginModule

Use the InternalLoginModule to authenticate against a user repository defined as a file on the file system. This is the default authentication mechanism for all webMethods suite products.

In case of successful authentication, the InternalLoginModule provides a user repository manager. It also creates a SagUserPrincipal object, and, optionally, a set of SagGroupPrincipal objects.

Parameter	Description
domain	Optional. String. Domain name to use for authentication. Applicable if the domain usage is activated for the InternalLoginModule.

Parameter	Description
internal Repository	Path to the internal user repository file.
group RepositoryPath	Optional. Path to the internal group repository file.
create_group_ principal	Optional. Whether to create group principals based on he information contained in groupRepositoryPath and attach the principals to the subject. Valid values are true or false (default).

The user-defined repository files must comply with this format:

```
*
* Default test repository for INTERNAL based authentication
*
* Copyright (c) 2001 - 2013 Software AG, Darmstadt, Germany and/or Software AG USA,
* Inc., Reston, VA, United States of America, and/or their licensors.
* All rights reserved.
version:3.0
*
*
user:username:$6a$kMpE+PvDv83zjcQe6fk7rWEiK80V73qoy9OZzr
0J4p4W3K1g9x1w2zEadkEjL2OLm1cozDfKJD7ZJckE3AysKw==
*
```

The group repository files must comply with this format:

```
*
*
* Default test repository for INTERNAL based authentication
*
* Copyright (c) 2001 - 2013 Software AG, Darmstadt, Germany and/or Software AG USA,
* Inc., Reston, VA, United States of America, and/or their licensors.
* All rights reserved.
version:3.0
*
*
admin:1:administrator,user2
testadmin:2:user2
*
```

The following sample outlines the INTERNAL mode of the InternalLoginModule and the corresponding configuration included in a login context of a JAAS configuration file.

```
LoginINTERNAL {
   com.softwareag.security.jaas.login.internal.InternalLoginModule required domain=
                                                  logCallback=true
        create_group_principal=true
        internalRepository="/tmp/myrepo/internalUserRepo"
        groupRepositoryPath="/tmp/myrepo/internalGroupRepo";
};
```

LDAPLoginModule

Use the LDAPLoginModule to authenticate users against an external directory. You can define your JAAS configuration to access information from an external directory if your site uses one of these external directories for user and group information:

- Lightweight Directory Access Protocol (LDAP)
- Active Directory acting as an LDAP server
- JAAS Configuration Properties

The following table outlines the JAAS configuration properties for all LDAP connections.

Parameter	Description
enabled	Optional. Whether to load the JAAS configuration. Valid values are true (default) or false. This parameter relates to dynamic configuration and should be set in the dynamic configuration property file. It should not be set in the JAAS configuration, and will have no effect if it is set there.
alias	Optional. Alias of the LDAP configuration entry. If not specified, it is set to match the url parameter. A valid value is any string of characters. The default is empty.
url	Required. URL to the LDAP server. If you want to use an SSL connection to the LDAP server, the URL should start with `ldaps`, and you should provide truststore and/or keystore parameters. The expected format is: ldap://*host* :*port* " or ldaps://*host* :*port* . If the URL points to IPv6 IP (not domain name), it must be enclosed in square brackets (for example, alias=ldap://[::1]:389).
domain	Optional. String. Domain name to use for authentication. Applicable if the domain concept is activated for the LDAPLoginModule. This parameter relates only to JAAS and should be set in the jaas.config file as a property of the LDAPLoginModule. It should not be set in the dynamic configuration property file, and will have no effect if it is set there.
applyDomain	Optional. Whether to apply domain when returning group information for the user. Valid values are true or false (default).

Parameter	Description
	This parameter relates only to JAAS and should be set in the jaas.config file as a property of the LDAPLoginModule. It should not be set in the dynamic configuration property file, and will have no effect if it is set there.
prin	Required if noPrinIsAnonymous is set to false; otherwise, do not specify this parameter. Distinguished name (DN) of the technical user that connects to the LDAP server if anonymous access to the LDAP server is not allowed.
noPrinIs Anonymous	Optional. When prin is not defined, specifies what credentials are used for LDAP server authentication. Valid values are: ■ true (default). The connection to the LDAP server is done anonymously. ■ false. The real user credentials of the user that connects to the LDAP server are also used for LDAP authentication. The LDAPLoginModule will need the complete DN for the user or activation of the useaf, dnprefix, dnsuffix parameters to be able to construct a proper user DN.
cred	Required if noPrinIsAnonymous is set to false; otherwise, do not specify this parameter. Password of the technical user that connects to the LDAP server. You use it with the prin parameter. A valid value is any string of characters.
credHandle	Can use instead of cred. Handles passman storage for technical user passwords. When a login is successful, cred is placed in passman.
timeout	Maximum time in milliseconds to spend for an LDAP operation. Default is 5000.
useaf	Optional. Boolean. Whether to use affixes (dnprefix and dnsuffix). Use the affixes for an easier construction of user DNs with less errors. Valid values are true or false (default).
dnprefix	Optional. String. Prefix to attach to the user name when performing operations on the LDAP server. To use this parameter, set useaf to true. A valid value is any string of characters.
dnsuffix	Optional. String. Suffix to attach to the user name when performing operations on the LDAP server. To use this

Parameter	Description
	parameter, set useaf to true. A valid value is any string of characters.
usecaching	Optional. Boolean. Whether the LDAP framework caches users and/or groups. Valid values are true (default) or false.
poolmin	Minimum number of objects to keep in the cache.
poolmax	Maximum number of objects to keep in the cache.
mattr	Optional. The LDAPLoginModule uses this parameter when performing member-search operations. The meaning of this parameter depends on the value of memberinfoingroups. If memberinfoingroups is set to true, the mattr parameter points from a group to the users that are members of this group. If memberinfoingroups is set to false, the mattr parameter points from a user entry to the groups that the user is a member of. A valid value is any string of characters. Default is memberOf.
memberinfoin groups	Optional. Boolean. Whether the login module searches users in a group or groups in a user. You can use it only if the mattr parameter is applied to users or groups. Valid values are true or false (default).
createGroups	Optional. Boolean. Whether to extract the groups of the logged-in user from the LDAP server. Valid values are true (default) or false. This parameter relates only to JAAS and should be set in the jaas.config file as a property of the LDAPLoginModule. It should not be set in the dynamic configuration property file, and will have no effect if it is set there.
createGroup Properties	Whether group properties should be populated to SagGroupPrincipal. Valid values are true or false (default). This parameter relates only to JAAS and should be set in the jaas.config file as a property of the LDAPLoginModule. It should not be set in the dynamic configuration property file, and will have no effect if it is set there.
createUser Properties	Whether user properties should be populated to SagUserPrincipal. Valid values are true or false (default). This parameter relates only to JAAS and should be set in the jaas.config file as a property of the LDAPLoginModule. It should

Parameter	Description
	not be set in the dynamic configuration property file, and will have no effect if it is set there.
uidprop	Optional. LDAP user name attribute. Default is CN.
gidprop	Optional. LDAP group attribute. A valid value is any string of characters. Default is CN.
grourootdn	Optional. Location from which to start searches for groups. A valid value is any string of characters.
groupobjclass	Optional. Specifies that the found object is a group. The login module uses this parameter when searching for groups. Default is group.
userrootdn	Optional. Location to search for users. A valid value is any string of characters.
personobjclass	Optional. Specifies that the found object is a person. The login module uses this parameter when searching for users. Default is person.
truststoreUrl	URL of the truststore to use if an SSL connection is required.
truststore Password	Password for the truststore if an SSL connection is required.
truststoreType	Type of truststore to use if an SSL connection is required.
keystoreUrl	URL of the keystore to use if an SSL connection is required.
keystore Password	Password for the keystore if an SSL connection is required.
keystoreType	Type of keystore to use if an SSL connection is required.
recursive SearchDepth	Amount of time to try when resolving nested groups (that is, a group that is a member of another group). The default is 0, which means no nested groups are resolved.
useFQDNFor Auth	Optional. Whether to try to log in with the complete name. This is supported only by Microsoft AD. Usually LDAP login

Parameter	Description
	module uses the user name or the complete DN of the user to log in. Valid values are true or false (default). If set to true, the LDAPLoginModule tries to login with DOMAIN*user_name* and password. This parameter relates only to JAAS and should be set in the jaas.config file as a property of the LDAPLoginModule. It should not be set in the dynamic configuration property file, and will have no effect if it is set there.

The following sample outlines the corresponding configuration included in a login context of a JAAS configuration file.

```
ExampleRealm {
  com.softwareag.security.sin.is.ldap.lm.LDAPLoginModule sufficient    alias="name1";
    com.softwareag.security.sin.is.ldap.lm.LDAPLoginModule sufficient
    alias="name2";
  com.softwareag.security.sin.is.ldap.lm.LDAPLoginModule sufficient;
  com.softwareag.security.sin.is.ldap.lm.LDAPLoginModule required
    alias="name3"
    url="ldap://localhost:389"
    prin="CN=sectest,OU=user,dc=example,dc=org"
    cred="******"
    useaf="true"
    dnprefix="CN="
    dnsuffix=",OU=user,dc=example,dc=org"
    usecaching="false"
    mattr="roleoccupant"
    memberinfoingroups=false
    creategroups=true
    gidprop="CN"
    grouprootdn="OU=Groups,dc=example,dc=org"
    groupobjclass="organizationalRole"
    personobjclass="organizationalPerson";
};
```

SAMLAssertValidatorLoginModule

Use SAMLAssertValidatorLoginModule to validate the delegation ticket issued from SAMLAssertIssuerLoginModule. You can use it for both SAML 1.1 and SAML 2 assertion validation.

The following sample outlines SAMLAssertValidatorLoginModule and the corresponding configuration included in a login context of a JAAS configuration file. The following login context is in the default jaas.config file that comes with Software AG Runtime.

```
/** Login context used in Common Platform for a default authentication **/
Default {
    // SSOS login module for SAML signed assertion validation
    com.softwareag.security.idp.saml.lm.SAMLAssertValidatorLoginModule sufficient;
    // Internal repository login module (java based)
         com.softwareag.security.jaas.login.internal.InternalLoginModule required
        template_section=INTERNAL
        logCallback=true
```

```
        internalRepository="C:/softwareag/common/conf/users.txt"
                    create_group_principal=true
        groupRepositoryPath="C:/softwareag/common/conf/groups.txt";};
```

SAMLAssertIssuerLoginModule

Use SAMLAssertIssuerLoginModule to issue a SAML1.1 or SAML 2 assertion as a delegation ticket among Software AG products.

You can only use the SAMLAssertIssuerLoginModule in a chain of login modules. Using this login module on its own, in a separate login context, is not possible, because it is the other modules in a given login context that perform the actual authentication of the user. When the authentication is successful, SAMLAssertIssuerLoginModule issues a SAML assertion where the fully qualified name of the authenticated user is part of the Subject of the AuthenticationStatement attribute of the SAML 1.1 assertion and the SubjectConfirmation attribute of the SAML 2 assertion. Optionally, the assertion contains a list of groups (where such are available) as part of the AttributeStatement attribute of the SAML assertion.

The SAMLAssertIssuerLoginModule has a single parameter that you set in the JAAS configuration.

Parameter	Description
forceSaml Version	Optional. Defines which SAML assertion version to use to issue the delegation token. Valid values are 1.1 or 2.0 (default).

The following sample excerpt outlines SAMLAssertIssuerLoginModule and the corresponding configuration included in a login context of a JAAS configuration file. First, InternalLoginModule authenticates the user. If the authentication is successful, SAMLAssertIssuerLoginModule issues a SAML 1.1 assertion to use as a delegation ticket.

```
/** Login Configuration for the SAMLAssertIssuerLoginModule. **/
SAMLIssuerRealm {
    // Internal repository login module (java based)
       com.softwareag.security.jaas.login.internal.InternalLoginModule required
        template_section=INTERNAL
        logCallback=true
        internalRepository="C:/softwareag/common/conf/users.txt"
                    create_group_principal=true
        groupRepositoryPath="C:/softwareag/common/conf/groups.txt";
    // SSOS login module for SAML 1.1 signed assertion issuance
    com.softwareag.security.idp.saml.lm.SAMLAssertIssuerLoginModule sufficient
                    forceSamlVersion="1.1";
};
```

JMXDelegatedAuthLoginModule

Use JMXDelegatedAuthLoginModule to validate the delegation ticket issued from SAMLAssertIssuerLoginModule or directly from the SSO service. You can use it for both SAML 1.1 and SAML 2 assertion validation. The purpose of this login module is to

support the JMX delegation mechanism. The login module gets a delegation ticket from the password field of the supplied credentials.

The following sample outlines JMXDelegatedAuthLoginModule and the corresponding configuration included in a login context of a JAAS configuration file. The following login context is in the default jaas.config file that comes with Software AG Runtime.

```
/*
 * Login context, used in Common Platform for management channel.
 */
PlatformManagement {
    // SSOS login module for SAML signed assertion validation
    // used for delegated authentication only for JMX
     com.softwareag.security.idp.saml.lm.JMXDelegatedAuthLoginModule sufficient;
      // Internal repository login module (java based)
     com.softwareag.security.jaas.login.internal.InternalLoginModule required
        template_section=INTERNAL
        logCallback=true
        internalRepository="C:/softwareag/conf/users.txt";
};
```

ServletHeaderLoginModule

Use ServletHeaderLoginModule to extract information from an HttpServletRequest which is sent to the login module as part of the SagCredentials. The login module extracts the X.509 certificate chain or SAML artifacts, which are received as a result of an HTTPS with ClientAuthentication against a web server. The login module enters this information into the SagCredentials and makes it available to other login modules used in the login context of a JAAS configuration file. Optionally, the login module can extract more information, such as user names and passwords.

The following table outlines the parameters of ServletHeaderLoginModule.

Parameter	Description
saml_artifact_ prop_name	Optional. Name of the SAML artifact property. Default is SAMLArt.
netegrity_ siteminder_ prop_name	Optional. Defines the name of the Netegrity SiteMinder property. Default is SM_USER.

The following sample outlines ServletHeaderLoginModule and the corresponding configuration that is included in a login context of a JAAS configuration file.

```
/** Login Configuration for the ServletHeaderLoginModule. **/
ServletHeaderLogin {
    com.softwareag.security.jaas.login.modules.ServletHeaderLoginModule optional;
};
```

SimpleNameMappingLoginModule

Use SimpleNameMappingLoginModule to map a user name that is in the sharedState or CallbackHandler to another user name, which is for example in a different user repository. The login module sends the result in the sharedState map. Depending on the parameters you include in the JAAS configuration file, you can provide different mapping modes with the login module. The properties mapping mode is based on a Java properties file. The regular expression mapping mode is based on the java.util.regex package. To enable a mapping mode you must use the corresponding configuration parameter in the JAAS configuration. You cannot use both mapping modes at the same time.

For more sophisticated mapping method, you can sub-class SimpleNameMappingLoginModule. Using the following sample excerpt, you can rework the method as explained. You can use the context parameter to define the target context for which the mapping is performed. The SagCredentials are sent by the application which calls the login module and therefore, must not be modified. You set the values of the super class variables using the mapName method and mapPassword method, if applicable.

```
protected mapName(String context, SagCredentials credentials, Map options)
throws SagGeneralSecurityException
```

The following table outlines the parameters of SimpleNameMappingLoginModule.

Parameter	Description
user_mapping_url	Required if you use properties file mapping. URL of the Java properties file that contains the mapping information.
user_mapping_regex	Required if you use regular expression mapping. Regular expression to use to collect the user name from the input name.
user_mapping_matchgroup	Optional. Regular expression group that is used for the results of the regular expression. Default is 1.

Examples are shown below.

- If you add this login module to the stack:

  ```
  fcom.softwareag.security.jaas.login.modules.SimpleNameMappingLoginModule required
       user_mapping_url=file://path/to/mapping_user.properties
  ```

  The mapping_user.properties file contains these entries:

  ```
  testclient=Test Client
  testclient.password=secret1
  ```

  If you login with user name testclient, the login modules after SimpleNameMappingLoginModule will receive user name Test Client and password secret1 as credentials.

- If you add this login module to the stack:

```
com.softwareag.security.jaas.login.modules.SimpleNameMappingLoginModule required
  user_mapping_regex="CN=(\\w*),(.*)"
```

  If you login with user name CN=Client1, OU=R&D, O=RSUBJET, C=DE the login modules after SimpleNameMappingLoginModule will receive user name Client1 as credentials.

- If you add this login module to the stack:

```
com.softwareag.security.jaas.login.modules.SimpleNameMappingLoginModule required
  user_mapping_regex="CN=(\\w*),(.*)"
  user_mapping_matchgroup="3"
```

  If you login with user name CN=Client1, OU=R&D, O=RSUBJET, C=DE the login modules after SimpleNameMappingLoginModules will receive user name null as credentials.

X509CertificateLoginModule

Use X509CertificateLoginModule to verify one or more than one X.509 certificate. The login module builds all chains of trust and at least one chain must end at the Trust Anchor. All certificates in the chain are verified according to the Public Key Infrastructure extensions (PKIX). The module checks the statuses of the certificates against Certificate Revocation Lists (CRLs). It can import missing certificates from PKCS#7 files. To get the CRL, the validation of the login module supports CRL distribution point (CRL DP). To enable CRL DP, you can set the value of the Java system property com.sun.security.enableCRLDP to true. The login module also provides direct trust. This means that the module checks whether the end entity certificate is part of the truststore. If it is, direct trust is created and further CRL checks are disabled.

The following table outlines the parameters of the X509CertificateLoginModule. The parameters allow you to extend the login module functionality and plug in other certificate validation methods in it.

Parameter	Description
truststore_url	URL of the keystore that contains the Trust Anchors. This is the RootCA or certificate authority (CA) certificates that are trusted.
truststore_ password	Password of the trust keystore.
truststore_type	Optional. Type of the trust keystore. Valid values are PKCS7, PKCS12, or JKS (default).
check_crl_status	Boolean. Valid values are:

Parameter	Description
	■ true. The status of the end entity certificate is checked against a URL. In this case, the crl_url parameter must be set. ■ false (default). The login module is set to use direct trust.
crl_url	Required when the check_crl_status is set to true. Defines the URLs of the CRL for the end entity certificate. The URLs are separated by a space.
overwrite_ username	Optional. Boolean. Valid values are: ■ true (default). The user name is overwritten with the certificate subject distinguished name (DN). ■ false. The module accomplishes only validation of the certificates.
additional_ certificates_ container_url	Optional. URL of the container of additional certificates.
additional_ certificates_ container_type	Optional. Type of the container of additional certificates. Valid values are PKCS7, PKCS12, or JKS (default).
additional_ certificates_ container_ password	Required when the additional_certificates_container_type parameter is set to JKS or PKCS12. Password of the certificate container.

The following sample outlines X509CertificateLoginModule and the corresponding configuration that is included in a login context of a JAAS configuration file.
The example also shows how the login context reads crl_url, truststore_url, and truststore_password from the Java system parameters. Note that every Java system parameter that is included in the JAAS configuration file must have a value that differs from NULL or the empty string. Failure to do so may cause an exception on the system.

```
/** Login Configuration for the X509CertificateLoginModule **/
X509Login {
    com.softwareag.security.jaas.login.modules.X509CertificateLoginModule required
        check_crl_status=true
        crl_url="${com.softwareag.security.crl.url}"
        truststore_url="${com.softwareag.security.truststore.url}"
        truststore_password="${com.softwareag.security.truststore.password}"
        truststore_type=jks
        overwrite_username=false
        additional_certificates_container_url=
                "${com.softwareag.security.certificate.container.url}"
        additional_certificates_container_type="jks"
```

```
        additional_certificates_container_password=
                "${com.softwareag.security.certificate.container.password}";
};
```

SAMLArtifactLoginModule

Use SAMLArtifactLoginModule to verify credentials received as SAML artifacts. The module uses the opensaml library and supports SAML version 1.1. It sends a request and validates the SAML artifact against a SAML endpoint, which is the authority issuer of the artifact. The authentication is successful only if the endpoint validates the SAML artifact successfully. The result of the validation is a SAML response that contains information about the owner of the artifact. A part of this response is the user name. If configured in the JAAS configuration file, the login module can overwrite the user name in the SagUserPrincipal with the one that is received in the SAML response.

The following table outlines the parameters of SAMLArtifactLoginModule.

Parameter	Description
saml_identity_ provider_url	URL of the SAML authority that validates the artifact.
overwrite_ username	Optional. Boolean. Whether to overwrite the user name with the one that is received in the SAML artifact validation process. Valid values are true (default) or false.

The following sample outlines SAMLArtifactLoginModule and the corresponding configuration that is included in a login context of a JAAS configuration file. In this example, the login context reads the saml_identity_provider_url parameter from the Java system parameters. Note that every Java system parameter that is included in the JAAS configuration file must have a value that differs from NULL or empty string. Failure to do so may cause an exception on the system.

```
/** Login Configuration for the SAMLArtifactLoginModule **/
SAMLArtifactLogin {
    com.softwareag.security.jaas.login.modules.SAMLArtifactLoginModule required
        saml_identity_provider_url="${com.sample.security.saml.samlendpoint}"
        overwrite_username=true;
};
```

RoleLoginModule

RoleLoginModule provides authorization information using the roles/permissions storage. The module is implemented according to the JAAS standards. The current user that is already successfully authenticated by other login modules from the chain, is searched in the storage by the fully qualified name. Also, if any of the previous login modules in the chain provides group membership of the user, this login module looks in the storage for the groups and concatenates permissions assigned to the group to the user's permissions. The login module updates already existing SagUserPrincipal

with the permissions assigned to the current user (direclty assigned or coming from the groups on which is member). Additionally, SagRolePrincipal is created for each role on which the user is member and all of those SagPrincipal objects are attached to the Subject.

Note: Permissions are added as properties of SagUserPrincipal with key name "permissions."

This module recognizes the configuration options below.

Parameter	Description
provider_class= my.provider. class	Optional. Full class name of the role provider to use. Default is FileBasedAuthzStoreImpl.
storage_location= "C:/tmp/ roles.txt"	Location of the roles storage. For FileBasedAuthzStoreImpl, that is the full path to the roles file.

A sample configuration is shown below.

```
Default {
   // SSOS login module for SAML signed assertion validation
   com.softwareag.security.idp.saml.lm.SAMLAssertValidatorLoginModule sufficient;
   // Internal repository login module (java based)
   com.softwareag.security.jaas.login.internal.InternalLoginModule required
       template_section=INTERNAL
       logCallback=true
       internalRepository="C:/SoftwareAG/conf/users.txt"
       create_group_principal=true
       groupRepositoryPath="C:/SoftwareAG/conf/groups.txt";
   // Role repository login module
   com.softwareag.security.authz.store.jaas.login.RoleLoginModule optional
       storage_location="C:/SoftwareAG/conf/roles.txt";
  };
```

4 Working with Web Services

Configuring Web Services Stack

Configuring the Web Services Stack Runtime

The following table shows which files you use to configure the Web Services Stack runtime.

File	Use to configure
axis2.xml	Client side and server side of all deployed web services. The axis2.xml file is a configuration file provided by the Apache Software Foundation. For more information about the Axis2 parameters in this file, see the Axis2 Configuration Guide.
module.xml	Specific modules.
services.xml	Specific web services.

You configure Web Services Stack as an integrated component of Software AG Runtime. When Software AG Runtime is started, Web Services Stack uses the runtime configuration shown in the following table.

Files to Configure	Location in *Software AG_directory*/profiles/CTP/workspace/wsstack/repository
axis2.xml	/conf
module.xml	/modules
services.xml	/services

The module.xml and services.xml files are stored in the META-INF subdirectory within the module archive and the service archive, respectively, in *Software AG_directory*/profiles/CTP/workspace/wsstack/repository.

Configuring the axis2.xml File

Web Services Stack uses the parameters listed below in the axis2.xml file. The default values for the parameters are set on the server side of the Axis2 configuration. If you want to change the default value for a parameter, add the parameter to the axis2.xml file and provide the new value.

Important: The axis2.xml file contains important information such as the user name and password to use to log in to the Web Services Stack administration console. Change the default credentials to protect access to the axis2.xml file.

Parameter	Description
include WrappedTypes Declaration	Whether to include message-wrapper elements in the WSDL XSD schema. Axis2 processes an RPC-style WSDL definition and automatically creates a wrapper element and type definition for each message. Axis2 then processes internally any request or response as if it is in a document style with an element declaration for each message. Valid values are: ■ false - Axis2 creates a copy of the WSDL definition when processing the message types and modifies the copy instead of the original WSDL document. ■ true (default) - Axis2 creates the web service instance and automatically adds the auto-generated types to the XSD of the original WSDL definition.
enableWSDL Validation	Whether to validate WSDL documents against external resources. Valid values are false (default) and true.

enableSoap Validation	Whether to validate SOAP messages. Valid values are:
	■ false (default) - when Axis2 client side and server side exchange SOAP messages, the messages are not automatically validated if they comply with the SOAP specification.
	■ true - the SOAP validation can be enabled both on the server side and on the client side. On the server side you can enable the SOAP validation at these levels:
	  ■ Globally - set the parameter in the axis2.xml file.
	  ■ For a specific service group - set the parameter inside a ServiceGroup tag in the services.xml file.
	  ■ For a specific service - set the parameter inside a Service tag in the services.xml file.
	  ■ For a specific operation - set the parameter inside an Operation tag in the services.xml file.
	  ■ For a specific request - set the parameter programatically to MessageContext.
	■ On the client side you can enable SOAP validation at these levels:

Parameter	Description
	■ Globally - set the parameter in the axis2.xml file. ■ For operations that expect large SOAP messages - call programmatically using `Options.setProperty("disableSoapValidation",Boolean.TRUE).`
wsdl4jRegister Default Extension AttributeTypes	Whether to register default extension attribute types in the WSDL4J extension registry. Configuration is done on Input, Output and Fault WSDL elements using String type. Valid values are false (default) and true.

Since messages that Web Services Stack processes are not always in SOAP format, the message builders and message formatters provided by Axis2 are extended to ensure all messages are correctly converted. Below is Web Services Stack-specific information about the proprietary message builders and message formatters available in the axis2.xml configuration file.

The Web Services Stack axis2.xml file contains defined proprietary message builders for the text/xml, application/xml, and application/soap+xml content types to extend the default functionality provided by Axis2. The definitions are as follows:

```
<messageBuilders>
  <messageBuilder contentType="text/xml"
     class="com.softwareag.builders.RawXMLMessageBuilder" />
  <messageBuilder contentType="application/soap+xml"
     class="com.softwareag.builders.RawXMLMessageBuilder" /    >
  <messageBuilder contentType="application/xml    "
     class="com.softwareag.builders.RawXMLMessageBuilder" /    >
  <messageBuilder contentType="application/x-www-form-urlencoded    "
     class="org.apache.axis2.builder.XFormURLEncodedBuilder" /    >
  <messageBuilder contentType="multipart/form-data    "
     class="org.apache.axis2.builder.MultipartFormDataBuilder" /    >
</messageBuilders>
```

The Web Services Stack axis2.xml file has defined proprietary message formatters for the text/xml, application/xml, and application/soap+xml content types to extend the default functionality provided by Axis2. The definitions are as follows:

```
<messageFormatters>
  <messageFormatter contentType="text/xml"
     class="com.softwareag.formatters.RawXMLFormatter" />
  <messageFormatter contentType="application/xml"
     class="com.softwareag.formatters.RawXMLApplicationXMLFormatter" />
  <messageFormatter contentType="application/soap+xml"
     class="com.softwareag.formatters.RawXMLFormatter" />
  <messageFormatter contentType="application/x-www-form-urlencoded"
     class="org.apache.axis2.transport.http.XFormURLEncodedFormatter" />
  <messageFormatter contentType="multipart/form-data"
     class="org.apache.axis2.transport.http.MultipartFormDataFormatter" />
</messageFormatters>
```

Configuring the Client

In the axis2.xml file, set the securityConfigFile parameter to the absolute or relative path to the current working directory or the *repository path* /conf directory, or to the wsclientsec.properties file containing security-related information. For example:

```
<parameter name="securityConfigFile">wsclientsec.properties</parameter>
```

Configuring MTOM

Binary content often has to be re-encoded to be sent as text data with SOAP messages. MTOM enables you to selectively encode portions of the message, making it possible to send base64-encoded data as well as externally attached binary data. You can configure MTOM message encoding at the global level in the axis2.xml file or at the service or operation level in the services.xml file. Set the enableMTOM parameter to the one of these values:

- true - response is always MTOM-ized in case the message includes binary data of schema type xmime:base64Binary.
- false (default) - response is always non-MTOM-ized, even if the request is MTOM-ized.
- optional - response is MTOM-ized only if the request is MTOM-ized.

Configuring Web Service Security

Web Services Stack provides this set of security features:

- Message-level security, which secures message content.
- Transport-level security, which secures the communication channel. The most typical case of transport-level security is the use of HTTP transport over SSL.
- Client authentication.

Setting Up Message-Level Security

Web Services Stack provides symmetric and asymmetric message-level security between the web service client and the web service itself in both directions. The symmetric message security and the asymmetric message security are both part of the WS-Security specification. To apply message security, you have to make several configurations on both the client side and the server side.

You can use the Web Services Stack plug-in to Software AG Designer to create the needed security configuration. Security configurations in Web Services Stack are based on the WS-Security Policy specification. For more information, see *Web Services Stack Help*.

Configuring the Server Side

To configure the server side, you need a keystore file that contains the X.509 certificate of the server. The keystore file can also contain public keys.

Specifying Settings in the axis2.xml or services.xml File

1. Go to the *Software AG_directory* /profiles/CTP/workspace/wsstack/reposiroty/conf directory and open the axis2.xml file in a text editor.
2. You can enable keystore caching at the global level in this file by setting the cacheCryptoInstances parameter to true. Because the keystore configuration can be different for each message, the caching is executed per message. When a service is undeployed or stopped, cached keystores are removed.
3. When the sp:RequiredElements and sp:RequiredParts assertions are available in the security policy, they may not be resolved and validated properly. By default, when XPath expressions are handled in sp:RequiredElements assertion, the expressions are validated against the soap:Envelope element, instead of the soap:Header element. You can enable the change on the entire runtime in this file. Add these parameters:

```
<parameter name="enableRequiredElementsXPathCompatibility">true</parameter>
<parameter name="enableRequiredPartsValidation">true</parameter>
```

4. Open the services.xml file in a text editor.
5. You can enable keystore caching at the service, service group, or specific operation level in this file by setting the cacheCryptoInstances parameter to true. Because the keystore configuration can be different for each message, the caching is executed per message. When a service is undeployed or stopped, cached keystores are removed.
6. You can enable caching of initialized password callback handler classes to improve performance by setting the cachePasswordCallbackHandler parameter to true. The callback handler instance is always cached on the service instance and will be lost if the service is undeployed.
7. Depending on the security policy, the client may be required to send the token used for encryption signature within the message itself. In this case the server side does not need to have client certificates. However, Rampart still verifies whether the certificates are trustworthy, and it requires that at least the certificate of the issuer be present in the truststore. Therefore, you must instruct Rampart/WSS4J to use the client's certificate. Set the encryptionUser parameter to useReqSigCert.

   useReqSigCert is a special fictional encryption user recognized by the security module. In this case, the certificate that is used to verify your signature is also used for the encryption of the response. Therefore, it is possible to have only one configured encryption user for all clients that access the service.
8. When the sp:RequiredElements and sp:RequiredParts assertions are available in the security policy, they may not be resolved and validated properly. By default, when XPath expressions are handled in sp:RequiredElements assertion, the expressions are validated against the soap:Envelope element, instead of the soap:Header

element. You can enable the change on a specific web service in this file. Add these parameters:

```
<parameter name="enableRequiredElementsXPathCompatibility">true</parameter>
<parameter name="enableRequiredPartsValidation">true</parameter>
```

9. You can enable or disable the WS-I Basic Profile compliance mode for your web services by setting the wsiBSPCompliant parameter to true (default) or false. For more information about the usage of the WS-I Basic Security Profile compliance mode, see WS-I Basic Profile.

Specifying Settings in a Software AG Designer Web Service Client

When the sp:RequiredElements and sp:RequiredParts assertions are available in the security policy, they may not be resolved and validated properly. By default, when XPath expressions are handled in sp:RequiredElements assertion, the expressions are validated against the soap:Envelope element, instead of the soap:Header element.

You can enable sp:RequiredElements and sp:RequiredParts assertions in the business logic of a web service client using this code snippet:

```
IWSStaxClient client = SampleService;
client.getWSOptions().setProperty("enableRequiredElementsXPathCompatibility",
"true");
```

You can also set specific properties using Software AG Designer. For instructions, see *Web Services Stack Help*.

Example of Symmetric Binding Security Configuration in the services.xml File

You can configure keystore properties by adding a Rampart custom policy assertion to the services.xml file. In the code sample below, the value clientCertificate is in fact an example of an alias for a client's certificate that has to be stored into the server side keystore file. If you want to authenticate a client which uses a user name token, you have to provide a password callback handler class to validate the user name and the password received from the client. When you provide a password using the callback handler class, you make a check towards a given authentication module.

Note: This authentication mechanism applies to the user name security token and can be used in a similar way with other security tokens.

```
<wsp:Policy wsu:Id="UserDefined"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd">
      <wsp:ExactlyOne>
        <wsp:All>
          <sp:SymmetricBinding
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <wsp:Policy>
              <sp:ProtectionToken>
                <wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy">
                  <sp:X509Token
sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy/Include
Token/Never">
                    <wsp:Policy>
                      <sp:WssX509V3Token10/>
```

```
                      <sp:RequireDerivedKeys/>
                    </wsp:Policy>
                  </sp:X509Token>
                </wsp:Policy>
              </sp:ProtectionToken>
              <sp:AlgorithmSuite
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
                <wsp:Policy>
                  <sp:Basic128/>
                </wsp:Policy>
              </sp:AlgorithmSuite>
              <sp:Layout>
                <wsp:Policy>
                  <sp:Strict/>
                </wsp:Policy>
              </sp:Layout>
  <sp:IncludeTimestamp/>
            </wsp:Policy>
          </sp:SymmetricBinding>
          <sp:Wss10
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <sp:Policy>
              <sp:MustSupportRefKeyIdentifier/>
              <sp:MustSupportRefIssuerSerial/>
            </sp:Policy>
          </sp:Wss10>
          <sp:SignedSupportingTokens
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
            <wsp:Policy/>
          </sp:SignedSupportingTokens>
          <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
<ramp:user>service</ramp:user>
            <ramp:encryptionUser>clientCertificate</ramp:encryptionUser>

<ramp:passwordCallbackClass>com.softwareag.wsstack.pwcb.PasswordCallbackHandler
</ramp:passwordCallbackClass>
            <ramp:signatureCrypto>
              <ramp:crypto
provider="org.apache.ws.security.components.crypto.Merlin">
                <ramp:property
name="org.apache.ws.security.crypto.merlin.keystore.type">JKS</ramp:property>
                <ramp:property
name="org.apache.ws.security.crypto.merlin.file">service.jks</ramp:property>
                <ramp:property
name="org.apache.ws.security.crypto.merlin.keystore.password">openssl
</ramp:property>
              </ramp:crypto>
            </ramp:signatureCrypto>
            <ramp:encryptionCypto>
              <ramp:crypto
provider="org.apache.ws.security.components.crypto.Merlin">
                <ramp:property
name="org.apache.ws.security.crypto.merlin.keystore.type">JKS</ramp:property>
                <ramp:property
name="org.apache.ws.security.crypto.merlin.file">service.jks</ramp:property>
                <ramp:property
name="org.apache.ws.security.crypto.merlin.keystore.password">openssl
</ramp:property>
              </ramp:crypto>
            </ramp:encryptionCypto>
          </ramp:RampartConfig>
        </wsp:All>
      </wsp:ExactlyOne>
```

```
    </wsp:Policy>
```

Configuring the Client Side

When you use the client API to invoke web services that require security, you can specify security configuration settings through a properties file. The security configuration settings are loaded only if the web service policy contains security assertions.

Open the axis2.xml file in a text editor and set the securityConfigFile parameter to the file name and path to the custom properties file, as follows:

```
<parametername="securityConfigFile">D:/wsdev/SampleWSClient/wsclientsec.
properties</parameter>
```

If you do not define such a parameter, the client implementation looks for a wsclientsec.properties file in the current working directory. If a securityConfigFile parameter exists but the file specified cannot be found, you get an exception. If the parameter is not defined or a wsclientsec.properties file is not present in the current working directory, the configuration loading routine does not throw any exceptions.

Below is a list of the supported configuration parameters you can include in the custom security configuration properties file.

Parameter	Description
USERNAME	User name used by: ■ Web Services Stack UsernameToken function in the UsernameToken. ■ Web Services Stack signing function as the alias name in the keystore to get the user's certificate and the private key to perform signing. ■ Web Services Stack encryption function if `ENCRYPTION_USER` is not set.
ENCRYPTION_USER	Encryption user name. The encryption function uses the public key of this user certificate to encrypt the generated symmetric key. If this parameter is not set, then the encryption function uses the USERNAME parameter value to get the certificate.
USER_ CERTIFICATE_ ALIAS	Alias of the key pair in the keystore used to get the private key for the signature. If this parameter is not set, the signature function uses the USERNAME parameter value.
STS_ALIAS	STS alias used as an encryption user in case of a STS authentication.

Parameter	Description
POLICY_ VALIDATOR_ CLASS	Policy validator callback class responsible for validating the security header against the security policy. The default callback class is org.apache.rampart.PolicyBasedResultsValidator.
TIMESTAMP_ PRECISION_ IN_MS	Defines whether time stamp precision is in milliseconds. The setting concerns the Timestamp element that may be required/ included in the security header. This parameter is passed to wss4j WSSConfig. ■ true (default) - time stamp precision is in milliseconds. ■ false - time stamp precision is in the format yyyy-MM-dd'T'HH:mm:ss'Z'.
TIMESTAMP_ TTL	Time stamp time-to-live in seconds. Default value is 300. Valid value is any integer.
TIMESTAMP_ MAX_SKEW	Used in time stamp validation where the creation time stamp must not be later than current time plus the time skew in seconds. Default value is 300. Valid value is any integer.
PASSWORD_ CALLBACK_ HANDLER_ CLASS	Class that implements the javax.security.auth.callback.CallbackHandler callback interface. The security module loads the class and calls the callback method to get the password. The class must have a public default constructor with no parameters.
OPTIMIZE_ PARTS_ EXPRESSIONS	List of Xpath expressions that refer to nodes that must be MTOM-optimized. The configured value is a semicolon delimited list of Xpath expressions. **Important:** If this property is set, it overwrites any previously configured list of expressions and does not add them to the list.

OPTIMIZE_ PARTS_ NAMESPACES	List of namespaces taken into consideration when searching for the nodes that are to be MTOM-optimized. The optimizing utility must recognize the namespace prefixes in the OPTIMIZE_PARTS_EXPRESSIONS list to be able to retrieve correctly the nodes from the document. By default, the following namespaces are registered:
	`xmlns:ds=http://www.w3.org/2000/09/xmldsig#`
	`xmlns:xenc=http://www.w3.org/2001/04/xmlenc#`
	`xmlns:wsse=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-`

Parameter	Description

	`wssecurity-secext-1.0.xsd xmlns:wsu=http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd`
	The expected value for this property is a semicolon delimited list of XML namespace declarations, for example:
	`OPTIMIZE_PARTS_NAMESPACES= xmlns:ns1=http://myns1; xmlns:ns2=http://myns2`
	Note: If this property is set, it overwrites any previously configured list of namespaces and does not add them to the list.

CRYPTO_ PROVIDER_ SIGN	WSS4J-specific Crypto implementation to use to generate the signature. It can be set to either of the following: ■ org.apache.ws.security.components.crypto.Merlin (default) ■ org.apache.ws.security.components.crypto.BouncyCastle
KEYSTORE_ PROVIDER_ SIGN	Signature keystore provider. If not set the JVM uses the default keystore provider, usually Oracle. For more information, see the java.security.Provider Java doc.
KEYSTORE_ TYPE_SIGN	Signature keystore type. If not set, the JVM uses the default keystore type, usually JKS. For more information, see the java.security.KeyStore#getDefaultType() method Java doc.
KEYSTORE_ FILE_SIGN	Signature keystore file.
KEYSTORE_ PASSWORD_ SIGN	Signature keystore password.
CRYPTO_ PROVIDER_ ENCRYPT	WSS4J-specific Crypto implementation to use for encryption. It can be set to either of the following: ■ org.apache.ws.security.components.crypto.Merlin (default) ■ org.apache.ws.security.components.crypto.BouncyCastle
KEYSTORE_ PROVIDER_ ENCRYPT	Encryption keystore provider. If not set the JVM uses the default keystore provider, usually Oracle. For more information, see the java.security.Provider Java doc.

Parameter	Description
KEYSTORE_ TYPE_ ENCRYPT	Encryption keystore type. If not set, the JVM uses the default keystore type, usually JKS. For more information, see the java.security.Provider Java doc.
KEYSTORE_ FILE_ENCRYPT	Encryption keystore file.
KEYSTORE_ PASSWORD_ ENCRYPT	Encryption keystore password.
CRYPTO_ PROVIDER_STS	WSS4J-specific Crypto implementation to use for protection in case of a STS authentication. It can be set to either of the following: ■ org.apache.ws.security.components.crypto.Merlin (default) ■ org.apache.ws.security.components.crypto.BouncyCastle
KEYSTORE_ PROVIDER_STS	Keystore provider to use in case of a STS authentication. If not set the JVM uses the default keystore provider, usually Oracle. For more information, see the java.security.Provider Java doc.
KEYSTORE_ TYPE_STS	Keystore type to use in case of a STS authentication. If not set the JVM uses the default keystore type, usually JKS. For more information, see the java.security.KeyStore#getDefaultType() method javadocs.
KEYSTORE_ FILE_STS	Keystore file to use in case of a STS authentication.
KEYSTORE_ PASSWORD_ STS	Keystore password to use in case of a STS authentication.

The configuration loading routine puts all those entries in the client options. You can overwrite any of the parameters next time Rampart is to be executed. For example, all security parameters can be specified programmatically using the Web Services Stack client options:

```
//create the WS Stack client:IWSStaxClient client = ......
IWSOptions options =
client.getWSOptions();options.setProperty(WSClientConstants.KEYSTORE_PASSWORD_
SIGN,
"changeit");options.setProperty(WSClientConstants.KEYSTORE_FILE_SIGN,
"C:\\client.jks");//execute the clientclient.sendReceive(...);
```

The Rampart is afterwards configured through a Rampart assertion that is generated by the RampartConfigLoader handler. The Web Services Stack client takes care of engaging that handler if Rampart itself is engaged. The function of the RampartConfigHandler is basically to gather all the security configuration keys, build up the Rampart configuration assertion, and put it as a property in the message context options where Rampart can find it.

Setting Up Transport-Level Security

You can set up transport-level security as follows:

- Configure Software AG Runtime to use SSL at the server side.
- Configure SSL at the client side.
- Configure SSL with client authentication
- Configure HTTP basic authentication.

Configuring Software AG Runtime to Use SSL at the Server Side

You set up Software AG Runtime to use the HTTPS transport for web service communication by configuring an SSL connector.

Important: Normally when you use Axis 2 in a web container, you must define the connector in the container and in the axis2.xml file. Software AG Runtime automatically registers the transport listener for you based on the HTTPS connector. If you define the use of HTTPS transport in the services.xml file, do not define a transport listener in the axis2.xml file.

Go to the *Software AG_directory*/profiles/CTP/configuration/com.softwareag.platform.config.propsloader directory, open the com.softwareag.catalina.connector.https.pid-*port* .properties file, and set the properties below.

Property	Description
clientAuth	Whether to require a certificate from the client. Valid values are: ■ true - require a valid certificate chain from the client before accepting a connection. ■ want - request a client certificate chain, but do not fail if one is not presented. ■ false (default) - do not require a certificate chain.
sslProtocol	Version of SSL to use. The default is TLS.

Property	Description
SSLEnabled	Whether to enable SecureSocketLayer protocol. Valid values are true or falsel (default).
keystoreFile	Path to the keystore file that contains the server certificate to use to decrypt the requests and encrypt the responses.
keystorePass	Password that provides access to the server certificate. If you want to secure the password, replace keystorePass with @secure.keystorePass.
keystoreType	Type of keystore file to use for the server certificate. The default is JKS.
keyAlias	Alias that identifies the key pair in the keystore. If not specified, the first key found in the keystore is used.
algorithm	Certificate encoding algorithm to use.
port	TCP port number on which this connector should create a server socket and wait for incoming connections. If not specified, the value is 10011. If you install another Software AG Runtime, the installer calculates a new port for that installation that is not already in use.
scheme	Configured scheme for the SSL communication. Set the value to https.
enableLookups	When there are IP addresses that connect to the port (before putting data in logs, for example), Tomcat may try to reverse lookup the name of the IP. For example, for IP=127.0.0.1, reversed lookup is localhost and localhost is displayed in logs. Valid values are true or false (default).
secure	Set this property to true.
minSpareThreads	Number of request processing threads to create when this connector is first started. The default is 10.
maxSpareThreads	Maximum number of request processing threads to create. The default is 75.

Property	Description
maxThreads	Maximum number of request processing threads to create. The default is 200.
acceptCount	Maximum queue length for incoming connection requests when all possible request processing threads are in use. The default is 100.
maxHttpHeaderSize	Maximum size of the request and response HTTP header, specified in bytes. If not specified, this value is 4096 (4 KB).
disableUploadTimeout	Allows the use of a different, longer connection timeout in connectionUploadTimeout. If not specified, this value is true.
connectionUpload Timeout	Connection timeout, in milliseconds. The default is 300000 milliseconds (5 minutes).

Below is an example of an SSL connector configuration.

```
clientAuth=false
sslProtocol=TLS
SSLEnabled=true
keystoreFile=c:\my_store.jks
@secure.keystorePass=password
keystoreType=JKS
keyAlias=encryption_key_alias
algorithm=SunX509
scheme=https
enableLookups=false
secure=true
minSpareThreads=25
maxSpareThreads=75
maxThreads=150
acceptCount=100
maxHttpHeaderSize=8192
disableUploadTimeout=true
enabled=trueport=10011
alias=defaultHttps
server=SoftwareAG Runtime
description=Default HTTPS Connector
```

Note: The default value of the connector port is 10011. If you install another Software AG Runtime, the installer calculates a new port for that installation that is not already in use.

Configuring SSL at the Client Side

The client must send a request to the HTTPS endpoint using the port specified at the server side. You can configure SSL at the client side using either of the methods below.

- Set the properties in your security configuration file. You can configure this file as a parameter in the axis2.xml configuration file:

```
<parametername="securityConfigFile">your_client_security_config_file
path</parameter>
```

  For information on the axis2.xml configuration file, see "Configuring the axis2.xml File" on page 62.

  If you do not define a security configuration file, the client uses information in the wsclientsec.properties file in the current working directory.

- Use the Web Services Stack client API to set the required properties, as follows:

```
//create the WS Stack client:IWSStaxClient client = ......

IWSOptions options = client.getWSOptions();
options.setProperty(WSClientConstants.KEYSTORE_PASSWO RD_SIGN, "changeit");
options.setProperty(WSClientConstants.KEYSTORE_FILE_SIGN, "C:\\client.jks");
//execute the clientclient.sendReceive(...);
```

The table below shows the security properties at the client side that relate to the SSL configuration. For more information, see the JSSE Reference Guide.

Property	Description
KEYSTORE_SSL_ LOCATION	Keystore file to use for SSL authentication. This property corresponds to the JSSE javax.net.ssl.keyStore system property. You need only specify the keystore file if the remote SSL server requires client authentication.
SSL_KEYSTORE_ PASSWORD	Password to use to access the keystore file. This property corresponds to the JSSE javax.net.ssl.keyStorePassword system property.
SSL_KEYSTORE_TYPE	Type of the keystore file.
TRUSTSTORE_SSL_ LOCATION	Truststore file to use for SSL authentication. The client requires that the server's certificate is installed in this truststore and it is trusted. This property corresponds to the JSSE javax.net.ssl.trustStore system property. If the property is not set, the client uses *Java-home* lib/ security/jssecacerts and *Java-home* /lib/security/cacerts, in that order.
TRUSTSTORE_SSL_ PASSWORD	Password for the truststore file. This property corresponds to the javax.net.ssl.trustStorePassword system property.

Configuring SSL with Client Authentication

On the server side, you can configure the Software AG Web Server based on Apache Tomcat to use a client certificate to encrypt the transferred data using either of the methods below.

- Go to the *Software AG_directory*/profiles/CTP/configuration/com.softwareag.platform.config.propsloader directory and open the com.softwareag.catalina.connector.https.pid-*port* .propertiesfile. Set the clientAuth property to true, and set the keystore and truststore properties.
- Configure the truststore location of the Software AG Runtime by starting it with the corresponding Java system property. If the truststore properties are not set in your configuration, Software AG Web Server based on Apache Tomcat uses the default Java trusted authority keystore. Specify these options in the *Software AG_directory*/profiles/CTP/configuration/config.ini file and then start Software AG Runtime:

```
javax.net.ssl.trustStore=full_path_to_truststore.jks
javax.net.ssl.trustStorePassword=password
```

Use the settings below to configure the truststore properties in the HTTPS connector.

Property	Description
truststoreFile	Truststore file to use to validate client certificates.
truststorePass	Password to use to access the truststore. The default ils keystorePass. You can add @secure in front of truststorePass.
truststoreType	Add this property if you are using a different format for the truststore than for the keystore.

Below is an example connector configuration.

```
clientAuth=true
sslProtocol=TLS
SSLEnabled=true
keystoreFile=C:\my_key/truststore.jks
truststoreFile=C:\my_key/truststore.jks
truststorePass=password
truststoreType=type
enabled=true
port=10011
keystorePass=password
keyAlias=key_alias
scheme=https
enableLookups=false
secure=true
alias=defaultHttps
maxSpareThreads=75
maxThreads=150server=SoftwareAG-Runtime
keystoreType=JKS
disableUploadTimeout=true
description=Default HTTPS Connector
```

```
algorithm=SunX509
minSpareThreads=25
acceptCount=100
maxHttpHeaderSize=8192
```

On the client side, you can use a client certificate with the Web Services Stack client, although additional work is needed to use the Java 1.4 -compatible HTTP sender with Jakarta Commons HttpClient. To make Commons HttpClient use a client certificate for the encryption, you must register a new HTTPS socket factory since the default one does not handle the case with the client certificate. Commons HttpClient does not provide the appropriate socket factory implementation, but you can use AuthSSLProtocolSocketFactory in the commons-httpclient-contib package that is part of the commons-httpclient project. You can set this as follows:

```
IWSStaxClient client = ......
ProtocolSocketFactory socketactory =
new AuthSSLProtocolSocketFactory(new File("keystore.jks").toURL(),
"keystorePassword", new File("truststore.jks").toURL(),
"truststorePassword");
Protocol authhttps = new Protocol("https", socketactory, 8443);
client.getWSOptions().setProperty(HTTPConstants.CUSTOM_PROTOCOL_HANDLE, authhttps);
```

Configuring HTTP Basic Authentication

With basic HTTP authentication, the server asks the client to provide its credentials in an HTTP authorization header. The enforcement of the basic HTTP authentication request can be delegated to the servlet container or can be left to the Web Services Stack security module (that is, Rampart).

The Rampart security module validates the usage of basic HTTP authentication. Rampart does not authenticate the user credentials sent in the HTTP header and only asserts whether the credentials are available. To authenticate successfully, you can use JAAS integration in Web Services Stack (see “Configuring Client Authentication” on page 80).

To avoid malfunction of the functionality, Web Services Stack must be running inside a servlet container or a server such as Integration Server. This is required because Rampart must be able to interact with the actual transport layer by accessing the transport level credentials and sending authorization request in case the basic HTTP authentication header is missing.

To validate basic HTTP authentication, Rampart must be informed that the service is secured by WS-SecurityPolicy. The following code sample denotes the basic HTTP authentication requirement:

```
<service name="ExampleService" ...>...<wsp:Policy
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
  xmlns:sp="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd" wsu:Id="user">
  <wsp:ExactlyOne>
    <wsp:All>
      <sp:TransportBinding xmlns:sp="http://docs.oasis-open.org/ws-sx/
ws-securitypolicy/200702">
        <wsp:Policy>
          <sp:TransportToken>
            <wsp:Policy>
```

```
              <sp:HttpsToken>
                <wsp:Policy>
                  <sp:HttpBasicAuthentication />
                </wsp:Policy>
              </sp:HttpsToken>
            </wsp:Policy>
          </sp:TransportToken><sp:AlgorithmSuite>
        <wsp:Policy>
      <sp:Basic256 />
   </wsp:Policy>
     </sp:AlgorithmSuite>
            <sp:Layout>
             <wsp:Policy>
               <sp:Lax />
              </wsp:Policy>
            </sp:Layout>
          <sp:IncludeTimestamp />
        </wsp:Policy>
      </sp:TransportBinding>...
    </wsp:All>
  </wsp:ExactlyOne>
</wsp:policy>
</service>
```

The sp:HttpBasicAuthentication assertion can appear only inside an sp:HttpsToken assertion, which means that the server also requires the use of HTTPS transport. To use this feature, you must engage Rampart for your web service by adding these lines to the service descriptor in the services.xml file:

```
<service name="ExampleService" ...>...
  <module ref="rampart"/>
</service>
```

Add a policy that contains the sp:HttpBasicAuthentication assertion to your web service. Below is an example.

```
<service name="ExampleService" ...>...
  <sp:HttpsToken>
    <wsp:Policy>
      <sp:HttpBasicAuthentication />
    </wsp:Policy>
  </sp:HttpsToken>...
</service>
```

To configure your web service client to use HTTP basic authentication, supply the HttpTransportProperties.Authenticator object in your client Java code, and set the user name and password to wssuser. Set this configuration as an option of the web service client. Below is an example web service client implementation that uses HTTP basic authentication.

```
IWSStaxClient client =
(IWSStaxClient)WSClientFactory.newClient( WSClientConstants.STAX_WSCLIENT,
"C:/ut_asym_xpath.wsdl",  null, null, "C:/Software AG/WS-Stack/repository");
HttpTransportProperties.Authenticator auth =
new HttpTransportProperties.Authenticator();
auth.setUsername ("wssuser");auth.setPassword("password");
auth.setPreemptiveAuthentication (true);
IWSOptions options = client.getWSOptions();
options.setProperty(org.apache.axis2.transport.http.HTTPConstants.
AUTHENTICATE,auth);
```

Configuring Client Authentication

Web Services Stack provides a mechanism for authenticating clients in its runtime layer using the JAAS security framework. Security Infrastructure provides you with JAAS-based login modules for client authentication. When you log on using a JAAS login context, a javax.security.auth.Subject is produced by the logon security module. That subject contains Principals and credentials and is available to anyone on the execution chain through the message context.

Web Services Stack collects all available security credentials from the client request and populates them in Security Infrastructure SagCredentials (see "Defining the Login Modules" on page 35). After that, the logon process is performed in the policy validator implementation of Rampart.

Configuring JAAS

Before you can log on, you must configure JAAS. For instructions, see "Setting Up Security" on page 33.

Security Credentials

Web Services Stack offers two types of user credentials for authentication:

- Message-level credentials. Web Services Stack can extract these credentials from the SOAP security header. If you use UsernameToken with plain text password, it can extract a user name and password. If there are signed parts or elements in the message, it can extract the X509Certificate used for the signatures.
- Transport-level credentials - communication channel used for the message exchange; they are specific to the type of transport you use. Web Services Stack extracts these credentials from the HTTP(S) transport only. In the case of a basic HTTP authentication, it extracts the user name and password. In the case of a client certificate used for encryption of the transferred data, it extracts a client certificate chain.

Implementing Password Callback Handlers

User-implemented password callback handlers are used to:

- Retrieve passwords to be placed inside a UsernameToken that corresponds to a given user name.
- Retrieve passwords to access user private keys from a keystore. The keystore password itself is directly set in the Rampart configuration.
- Verify the password in the received UsernameToken.

The callback handlers can retrieve passwords from configuration files, databases, LDAP servers, or other application components that are used for user management, such as Security Infrastructure.

Web Services Stack has a predefined set of password callback handlers that facilitate different scenarios for retrieving passwords. You can use these handlers directly or you can develop your own password callback handlers from them. You can use the password callback handlers below.

com.softwareag.wsstack.pwcb.ConfigFilePasswordCallbackHandler

The password callback handler retrieves identifier-password pairs from a configuration file and then loads the pairs which can be used to find the needed password for a particular identifier. The configuration file must be in XML format and similar to the axis2.xml file.

You can provide a configuration file to the callback handler by specifying it in the web service archive. In the services.xml file, you add a PWCBConfigFile parameter, which is set to point to the configuration file resource on the service class path. The class path includes the service archive, the libraries which are in the service archive, the web application class path (all jar files in WEB-INF/lib and the WEB-INF/classes class folder) and so on.

```
<serviceGroup>
 <service name="Sample_Web_Service">
   <parameter  name="PWCBConfigFileLocation"> configuration_file_location
  </parameter>  ...
 </service>
</serviceGroup>
```

If you do not specify the configuration file resource, by default the callback handler searches for a resource with name users.xml in the service class path. If it is not available, a FileNotFoundException is thrown.

The same password callback handler is also available at the client side if there is no service archive. Then, presumably, the configuration file is users.xml and is searched on the class path of the client. Then it is loaded as a resource.

com.softwareag.wsstack.pwcb.LdapPasswordCallbackHandler

The password callback handler retrieves identifier-password pairs from an LDAP server and then loads the pairs which can be used to find the needed password for a particular identifier. To retrieve data from the server, you set the URL of the LDAP server as well as some more properties in the handler. These properties are passed to the handler in a common properties file.

You can provide a common properties file to the callback handler by specifying the location of the common properties file in the web service archive. In the services.xml file, you add a PWCBLDAPPropFile parameter, which is set to point to the location of the properties file. The location of the file can be any valid path from which the handler can load the file (for example, conf/my-ldap.properties).

```
<serviceGroup>
 <service name="Sample_Web_Service">
  <parameter name="PWCBLDAPPropFileLocation"> common_prop_file_location
  </parameter>...
 </service>
</serviceGroup>
```

If you do not specify a properties file in the services.xml file, the password callback handler is configured to use a default properties file (ldap.properties) from the root directory.

The file may be also placed in a Java archive (.jar file) which resides in the WEB-INF/lib (for example, pwcb-server.jar) or directly in WEB-INF/classes directory. If the password callback handler does not discover the properties file in a pre-set directory, or in the root directory of the web service archive, it searches for the file in a central location on the class path of the handler and loads the properties file as a resource. If this process is unsuccessful, a FileNotFoundException is thrown.

The same password callback handler is also available at the client side if there is no service archive. Then, presumably, the configuration file is ldap.properties and is searched on the class path of the client. Then it is loaded as a resource.

If you do not provide an explicit properties file in the services.xml file, the password callback handler is configured to use a default properties file (ldap.properties) from the root directory.

The file may be also placed in a Java archive (.jar file) that resides in the WEB-INF/lib (for example, pwcb-server.jar) or directly in the WEB-INF/classes directory. If the password callback handler does not discover the properties file in a pre-set directory, or in the root directory of the web service archive, it searches for the file in a central location on the class path of the handler and loads the properties file as a resource. If this process is unsuccessful, a FileNotFoundException is thrown.

The same password callback handler is also available at the client side if there is no service archive. Then, presumably, the configuration file is ldap.properties and is searched on the class path of the client. Then it is loaded as a resource.

Implementing Policy Validation Callbacks

The wsstack-jaas.jar module offers ready-to-use policy validator implementations that you can configure and use to log on. Below are examples implementations. To use one of the callbacks, specify policyValidatorCbClass in the Rampart policy assertion.

- com.softwareag.wsstack.jaas.callback.SimpleSINPolicyValidatorCallback. Attempts to log on with all available credentials (message-level credentials are with higher priority over transport-level credentials) against the JAAS logon context. Specify the login context name as a parameter under the key sin.jaas.login.context. The resulting JAAS login subject is available as a property of the message context under the key sin.jaas.subject.
- com.softwareag.wsstack.jaas.callback.ServletRequestLoginPolicyValidatorCallback. Attempts to log on using the servlet request resource populated in the SIN credentials list. Specify the login context name as a parameter under the key sin.jaas.login.context. The resulting JAAS logon subject is available as a property of the message context under the key sin.jaas.subject.
- com.softwareag.wsstack.jaas.callback.MultiLoginPolicyValidatorCallback. Attempts to log on first with transport-level credentials and then again with message-level credentials. Specify the login context name as a parameter under the key

sin.jaas.login.context. The name of the transport login context is available as a parameter under the key sin.jaas.transport.login.context (default WSS_Transport_IS) and for message-level credentials logging on under sin.jaas.msg.login.context (default WSS_Message_IS). The resulting subjects are respectively populated as properties of the message context under the keys sin.jaas.transport.subject and sin.jaas.msg.subject.

These policy validator callbacks extend the standard callback that is provided by Rampart. This means that all basic functionality for validating security policy conformation is still present.

Authenticating Web Services

When you expose a web service, you might want to authenticate the user that is executing the service (for example, via user name/pass word, Kerberos, or certificate). This section describes how to configure the service to perform this authentication. For information about the authentication steps listed here, see "Setting Up Security" on page 33.

Configure the JAAS configuration file (see "Setting Up Security" on page 33). Then configure a web service to do the following:

- Specify the policyValidatorCbClass in the Rampart configuration policy assertion. Below is sample code for the Rampart policy assertion with specified policyValidatorCbClass:

```
<ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
  <ramp:user>service</ramp:user>
  <ramp:encryptionUser>client</ramp:encryptionUser>
  <ramp:policyValidatorCbClass>com.softwareag.wsstack.jaas.callback
     .MultiLoginPolicyValidatorCallback </ramp:policyValidatorCbClass>
```

- Specify the login context name as a parameter on one of the web service levels (global level in axis2.xml, service group level in services.xml, service level in services.xml, operation level in services.xml, message level in services.xml).
- To detect any changes in the configuration, the built-in policy validators provided by Web Services Stack automatically refresh the JAAS configuration prior to each login attempt. Since the configuration is shared for the entire Java virtual machine instance, this detection results in increased synchronization wait time on the server side. To improve the performance, you can disable the automatic refresh feature by setting the autoRefreshJaasConfig parameter to false.

  The parameter can be set globally in the axis2.xml configuration file or locally in the services.xml service descriptor. The following excerpt outlines the configuration of the parameter:

```
<parameter name="autoRefreshJaasConfig">false</parameter>
```

About Configuring Message Transports

Web Services Stack supports sending and receiving messages over HTTP or HTTPS, TCP, JMS, or Mail. This section explains how to configure and activate or deactivate the transports supported by Web Services Stack.

Configuring HTTP and HTTPS Transport

By default, the HTTP transport is activated and the HTTPS transport is deactivated in Web Services Stack.

The Default HTTPS Connector value is used by the Software AG Common Platform to distinguish default connectors from other existing connectors, and is present by default in the predefined Software AG Runtime HTTPS connector definition. Make sure the description property is set to Default HTTPS Connector in at least one of your HTTPS connectors, or the Software AG Runtime configuration will be invalid or corrupted the next time you install or upgrade a product that uses Software AG Runtime.

If you disable a connector in Software AG Runtime, you must also disable the corresponding transport sender and receiver in the Web Services Stack axis2.xml file, or an error will occur in Web Services Stack.

Activating or Deactivating HTTP or HTTPS

1. Go to the *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/conf directory and open the axis2.xml file.
2. Comment out the sections that define the transport receiver and transport sender with name="http" or name="https":

```
<transportReceiver name="http" ... />
<transportSender name="http" ... />
<transportReceiver name="https" ... />
<transportSender name="https" ... />
```

3. Restart Web Services Stack.

   Restart the Software AG Runtime Windows Service.

Activating or Deactivating HTTP or HTTPS in Software AG Runtime

1. Go to the *Software AG_directory*/profiles/CTP/configuration/com.softwareag.platform.config.propsloader directory.
2. Open the file that defines the connector to activate or deactivate (for example, com.softwareag.catalina.connector.http.pid-*identifier* .properties).
3. Set the “enabled” property to true or false.
4. Save the properties file. The change will be automatically detected and Software AG Runtime will update itself; no restart is required.

Configuring TCP Transport

Activating TCP Transport on the Server Side

1. Activate TCP transport as follows:

   a. Go to the *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/conf directory and open the axis2.xml file.

   b. Uncomment the sections that define the transport receiver and transport sender with name="tcp":

   ```
   <transportReceiver name="tcp" ... />
   <transportSender name="tcp" ... />
   ```

   The only parameter required for the transport receiver is its port number. The suggested value is 6060.

2. Restart Web Services Stack.

3. Since the TCP transport has no application level headers (and no target endpoint URI), you need WS-Addressing to dispatch the service. WS-Addressing may not be enabled in the default Web Services Stack installation. Enable WS-Addressing as follows:

   a. Engage the WS-Addressing module globally by adding in the axis2.xml configuration file the following line:

   ```
   <module ref="addressing"/>
   ```

   b. Engage the WS-Addressing module on a <service> level. Engagement is for the service that is deployed on TCP transport.You can enable WS-Addressing in the services.xml configuration file by adding the following line:

   ```
   <service ...>
    <transports>
      <transport>tcp</transport>
    </transports>
    <module ref="addressing"/>
   ...
   </service>
   ```

   c. Enable WS-Addressing by using the Web Services Stack plug-in to Software AG Designer. To do so, select **Enable WS-Addressing** from the **Modules** list in the **Services** tab. For more information, see *Web Services Stack Help*.

4. If not explicitly configured, a web service is deployed over all activated transports in Web Services Stack. In this case, the web service is accessible at all enabled endpoints. You may, however, want to restrict a web service to be accessible only over TCP transport.

   a. Configure the web service's services.xml file by adding the following on the <service> level:

   ```
   <service ...>
     <transports>
       <transport>tcp</transport>
   ```

```
  </transports>
...
</service>
```

b. Use Web Services StackDesigner plug-in at deployment time. To do this, select **TCP Transport** from the list of transports in the **Services** tab.

> **Note:** Since TCP transport has no application level headers, and thus no target endpoint URI, you need WS-Addressing to dispatch the service. If WS-Addressing is not globally enabled, you have to enable it for the service.

Invoking a Web Service Over TCP Transport on the Client Side

1. Make sure the WS-Addressing module called addressing.mar exists in the /modules directory in the client's repository.
2. Uncomment the sections that define the transport receiver and transport sender with name="tcp" in the client's axis2.xml configuration file:

```
<transportReceiver name="tcp" ... />
<transportSender name="tcp" ... />
```

3. Engage globally the addressing.mar module in the client's axis2.xml file:

```
<module ref="addressing"/>
```

Activating JMS Transport

Activating JMS Transport on the Server Side

1. Go to the *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/conf directory and open the axis2.xml configuration file.
2. Uncomment the sections that define the transport receiver and transport sender with name="jms":

```
<transportReceiver name="jms" ... />
<transportSender name="jms" ... />
```

3. Define the custom connection factories. You can define custom connection factories as parameters under JMS transport receiver. They can be used by the services deployed over JMS transport. Refer to the axis2.xml configuration file to see the sample connection factories that the JMS transport receiver configuration includes.

> **Note:** One of the connection factories is named as the default for use by services that do not explicitly specify the connection factory they want to use in their services.xml configuration file.

Each connection factory specifies parameters for an initial naming factory class, a naming provider URL, and the JNDI name of an actual JMS connection factory. Web Services Stack can run with the default configuration of Apache ActiveMQ, if you

use it. In this case, you only have to uncomment the JMS transport receiver and JMS transport sender configuration in the axis2.xml file.

Note: You must always run the message broker before you start Web Services Stack.

Force Deployment Over JMS Transport Only

If not explicitly configured, a web service is deployed over all activated transports in Web Services Stack. However, you can restrict a web service to be deployed over JMS transport only. You can also specify the destination where the service listens for messages, as well as the name of the connection factory to be used. The service can use one of the connection factories defined within the JMS transport receiver in the axis2.xml configuration file.

Do one of the following:

- Configure the web service's services.xml file by adding the <transport>jms</transport> element:

```
<service ...>
  <transports>
    <transport>jms</transport>
  </transports>...
</service>
```

- Use the Web Services Stack plug-in to Software AG Designer at deployment time by selecting **JMS Transport** from the list of transports in the **Services** tab.

Specifying the Connection Factory Name

You can specify a name for the connection factory that the web service will use. This can be done by modifying the services.xml file or by using the Web Services Stack plug-in to Software AG Designer. The parameters that define the connection factory name are optional. If they are not specified, the service uses the default connection factory (named "default" in the configuration of the JMS transport receiver in the axis2.xml file) and listens for messages on a JMS queue by the same name as the name of the service.

You can specify the connection factory name through the services.xml file by adding the <parameter name> elements. The connection factory can be any of those defined in axis2.xml and the destination name can be anything. transport.jms.ConnectionFactory and myQueueConnectionFactory are sample parameter values.

```
<service ...>
  <transports>
    <transport>jms</transport>
  </transports>
  <parameter name="transport.jms.ConnectionFactory" locked="true">
    myQueueConnectionFactory</parameter>
  <parameter name="transport.jms.Destination" locked="true">
    dynamicQueues/TestQueue</parameter>
...
</service>
```

1. In the **Project Explorer** view, select the web service archive that will use the connection factory.
2. Click the **Services** tab.
3. Specify the connection factory. In the **Properties** section, click **Add**. Type transport.jms.ConnectionFactory in the **Name** field, and type myQueueConnectionFactory (or another connection factory defined in axis2.xml) in the **Value** field. Then click **OK**.
4. In the **Properties** section, click **Add**. Type transport.jms.Destination in the **Name** field, and type dynamicQueues/TestQueue (or other value of your choice) in the **Value** field. Then Click **OK**.

   The connection factory name is now set and visible in the **Services.xml** tab.

Invoking a Web Service Using JMS on the Client Side

1. Make sure the WS-Addressing module called addressing.mar exists in the /modules directory in the client's repository.
2. Uncomment the sections that define the transport receiver and transport sender with name="jms" in the client's axis2.xml configuration file:

```
<transportReceiver name="jms" ... />
<transportSender name="jms" ... />
```

3. Engage globally the addressing.mar module in the client's axis2.xml file.

```
<module ref="addressing"/>
```

Configuring Mail Transport

Setting Up Apache James Server

The activation of mail transport in Web Services Stack requires the open source SMTP and POP3 Apache Java Enterprise Mail Server (James) to transfer e-mail messages. After you have installed and configured your the Apache James server, you must create a mail account that represents the e-mail address of Web Services Stack. You can create additional accounts to correspond to different clients. For more information on configuring the Apache James mail server, see the Apache James documentation.

1. Install Apache James server as follows:
   a. Download the archive with the binary distribution of the Apache James mail server from the Apache James website.
   b. Extract the files from the downloaded archive to a JAMES_HOME directory of your choice.
   c. Start and stop the mail server once so that it unpacks its configuration files.
2. Open the configuration files for editing as follows:

a. Open a command prompt and to go the JAMES_HOME/bin directory.

b. Run run.bat to start the server, then use the CTRL+C command to stop the mail server.

c. Type the `ipconfig /all` command to check your network configuration.

3. Configure the DNS servers in the mail server as follows:

a. Open the config.xml file under the JAMES_HOME/apps/james/SAR-INF directory.

b. Find the tag dnsserver and enter the IP address of each DNS server from your network configuration as shown in the following example:

```
<dnsserver>
<servers>
<server>[DNS.Server.IP.address]</server>

<server>...</server>
</servers>
...</dnsserver>
```

c. Start the mail server again.

4. Create accounts in the mail server as follows:

a. Start the Apache James mail server. To do so, run the console command prompt, navigate to JAMES_HOME/bin directory and run `run.bat`.

b. Start the James Remote Manager Service. Run the console command prompt and type the following telnet command:

```
telnet localhost 4555
```

Port number 4555 is the default port, where the Remote Manager Service starts. It is configured in the James configuration file (JAMES_HOME/apps/james/SAR-INF/config.xml). If you have changed the default port number in a previous step, use the new value in the preceding command.

c. Log on the Remote Manager. You are prompted for the logon ID and password. They are configured in the James configuration file (JAMES_HOME/apps/james/SAR-INF/config.xml). The initial values are "root" for both the login ID and the password, unless you have changed them.

d. Create the account using the command `adduser username password`.

e. Type the command `adduser server wsstack`.

f. Exit the Remote Manager Service using the `quit` command.

After you have executed the commands in the command prompt, you get a result similar to the following one:

```
>telnet localhost 4555
JAMES Remote Administration Tool 2.3.1
Login id:root
Password:root
Welcome root.
HELP....
```

```
quit
Bye
```

Activating Mail Transport on the Server Side

1. Go to the *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/conf directory and open the axis2.xml file.
2. Find the contextRoot parameter. If it is commented out, uncomment it and make sure its value is wsstack:

```
<parameter name="contextRoot" locked="false">wsstack</parameter>
```

3. Uncomment the sections that define the transport receiver and the transport sender with name="mailto":

```
<transportReceiver name="mailto" … />
<transportSender name="mailto" … />
```

   The parameters under the transport receiver and the transport sender have default values; verify these values.

4. Set the values on the required parameters for the transport receiver.

Parameter	Description
mail.pop3.host	Host name (or IP address) for the machine that hosts the James mail server. If the server is running on the same machine as Web Services Stack, you can set the value to "If the server is running on the same machine as Web Services Stack, you can set the value to "localhost" or "127.0.0.1".
mail.pop3.user	User name of a user registered in the James mail server.
transport. mail.pop3. password	Password for the specified user name.
mail.store. protocol	Value must be "pop3".
transport.mail. replyTo Address	■ Supplies the endpoint reference for the response and represents the server email address. ■ Contains the user name specified in the mail.pop3.user parameter and the server name of the James mail server, separated by the @ sign. The server name is configured in the JAMES_HOME/apps/james/SAR-INF/config.xml configuration file. If you have not specified a different one, the initial value is "`localhost`".

Parameter	Description
transport. listener.interval	Interval, in milliseconds, at which to check the mail server for new messages. If you do not specify a value, the default is 3000 milliseconds.

Below is sample code that shows the usage of the required parameters for the transport receiver.

```
<transportReceiver name="mailto" class="org.apache.axis2.transport.mail.SimpleMailListener">
  <parameter name="mail.pop3.host">localhost</parameter>
  <parameter name="mail.pop3.user">server</parameter>
  <parameter name="transport.mail.pop3.password">wsstack</parameter>
  <parameter name="mail.store.protocol">pop3</parameter>
  <parameter name="transport.mail.replyToAddress">server@localhost</parameter>
  <parameter name="transport.listener.interval">3000</parameter>
</transportReceiver>
```

5. Set the values on the required parameters for the transport sender.

Parameter	Description
mail.smtp.host	Host name (or IP address) for the machine that hosts the James mail server. The value corresponds to the mail.pop3.host parameter under the transport receiver.
mail.smtp.user	Corresponds to the value of the mail.pop3.user parameter of the transport receiver.
transport.mail. smtp.password	Corresponds to the value of the transport.mail.pop3.password parameter of the transport receiver.
mail.smtp.from	Corresponds to the value of the mail.transport.replyToAddress parameter of the transport receiver.

Below is sample code that shows the usage of the required parameters for the transport sender.

```
<transportSender name="mailto" class="org.apache.axis2.transport.mail.
MailTransportSender">
  <parameter name="mail.smtp.host" locked="false">localhost</parameter>
  <parameter name="mail.smtp.user">server</parameter>
  <parameter name="transport.mail.smtp.password">wsstack</parameter>
  <parameter name="mail.smtp.from">server@localhost</parameter>
</transportSender>
```

Force Deployment Over Mail Transport Only

If not configured explicitly, a web service is deployed over all activated transports in Web Services Stack. If you want to restrict a web service to be deployed only over Mail transport, you must add this element in the web service's services.xml file:

```
<service ...>
 <transports>
   <transport>mailto</transport>
 </transports>...
</service>
```

Invoking a Web Service Over Mail Transport on the Client Side

In the client's axis2.xml configuration file, find and uncomment the sections that define the transport receiver and transport sender with `name="mailto"`. Check the parameters under the mail transport receiver and the mail transport sender. You must configure the user name, the password, and the e-mail address of a user registered in the James mail server. That user must be different from the one configured in Web Services Stack.

Below is sample code for client configuration with a user that is registered in the James mail server. The user name is "client" and the password is "pass".

```
<transportReceiver name="mailto" class="org.apache.axis2.transport.mail.SimpleMailListener">
  <parameter name="mail.pop3.host">localhost</parameter>
  <parameter name="mail.pop3.user">client</parameter>
  <parameter name="mail.store.protocol">pop3</parameter>
  <parameter name="transport.mail.pop3.password">pass</parameter >
  <parameter name="transport.mail.replyToAddress">client@localhost</parameter>
  <parameter name="transport.listener.interval">3000</parameter>
</transportReceiver>
<transportSender name="mailto" class="org.apache.axis2.transport.mail.MailTransportSender">
  <parameter name="mail.smtp.host">localhost</parameter>
  <parameter name="mail.smtp.user">client</parameter>
  <parameter name="transport.mail.smtp.password">pass</parameter >
  <parameter name="mail.smtp.from">client@localhost</parameter>
</transportSender>
```

Monitoring SOAP Messages

Web Services Stack comes with a SOAP monitor you can use to monitor SOAP messages that are exchanged between web service clients and web services running in Web Services Stack.

The SOAP monitor shows SOAP messages with the structure they have after they have passed all system phases in the Axis 2 engine. This means that the original SOAP messages sent by a user can be visually different but are semantically equal to the ones shown into the SOAP monitor. Examples of such a case are MTOM SOAP messages. SOAP monitor shows the binary data exchanged "by value" (included into the SOAP message itself). On the other hand, the original SOAP message has MIME parts in it.

For example, open TCPMon and extract the data of the exchanged message in binary format. For ease of use, only the part of the message related to the MTOM-ized binary data is shown:

```
<ns1:binaryData><xop:Include
href="cid:1.urn:uuid:EFF202258F699D83131220514272228@apache.org"
xmlns:xop="http://www.w3.org/2004/08/xop/include" /></ns1:binaryData>…--
MIMEBoundaryurn_uuid_EFF202258F699D83131220514272117Content-Type:
text/plainContent-Transfer-Encoding: binaryContent-ID:
<1.urn:uuid:EFF202258F699D83131220514272228@apache.org>text--
MIMEBoundaryurn_uuid_EFF202258F699D83131220514272117—
```

The binary data displayed by the SOAP monitor in the example above is shown below. The binary data is shown “by value," because it was already processed by the system phases of the Axis 2 engine.

```
<ns1:binaryData>dGV4dA==</ns1:binaryData>
```

For more information on the SOAP monitor configuration, see the Apache documentation.

The SOAP monitor is disabled by default.

Enabling the SOAP Monitor in the Web Services Stack

1. Go to the *Software AG_directory*\profiles\CTP\workspace\wsstack\repository\conf directory and open the axis2.xml file.
2. Engage the `soapmonitor` Axis2 module globally in the axis2.xml or for a service in the services.xml fileby adding this line:

   ```
   <module ref="soapmonitor"/>
   ```
3. Add a soapMonitorPort parameter, which defines the port to use for communication with the SOAP Monitor Applet

   ```
   <parameter name="soapMonitorPort">5001</parameter>
   ```

   Important: If you do not add this parameter, the SOAP Monitor servlet will not be available.
4. Restart Web Services Stack.
5. Go to http://*host* :*port* /wsstack/SOAPMonitor to start using the SOAP monitor.

Configuring Logging

Configuring Logging in Web Services Stack

Web Services Stack uses Journal Logging as a logging mechanism. The Journal Logging is delivered with the shared component bundle com.softwareag.sc.core and its configuration file is located in the *Software AG_directory*/profiles/CTP/configuration/logging directory in the log_config.xml file.

The Journal Logger is a wrapper around log4J and every Journal Logging logger wraps a standard log4J logger. For this reason, the Journal Logger component delivers log4J as part of its implementation. The Journal Logger configuration is a standard log4J configuration that sets up the underlying log4J library. If necessary, you can use log4J directly. You should add your log4J settings to the Journal Logger configuration file.

Basically, the format of the log_config.xml file is the same as the format of the log4J XML configuration. The Journal Logger contains several additional appenders than the standard log4J appenders.

To enable logging and configure the corresponding severity, open the log_config.xml file and edit this excerpt as follows:

```
<root>
   <level value="info" />
   <appender-ref ref="Platform.Console" />
    <appender-ref ref="Platform.RollingLogFile" />
</root>
```

Deploying Web Services Stack

Web Services Stack distributes the Bouncy Castle JCE provider. It is required by the security module (Rampart) for retrieving cryptographic algorithms implementation used in encryption and/or signing of messages.

The Bouncy Castle provider is added to the runtime list of Java security providers (when required for the first time).

The Bouncy Castle provider might not be available to other web application if Web Services Stack is deployed in a servlet container and the Bouncy Castle classes are loaded from the Web Services Stack web application classloader. After it is added to the global list of security providers, no other application running in the same virtual machine will be able to add it again. In this case, if the Bouncy Castle is required by other web application in the servlet container, place the Bouncy Castle JAR in a common/shared lib directory of the servlet container and ensure it is loaded from there and not by a web application classloader.

Note: If Web Services Stack is undeployed, it will take care of unregistering Bouncy Castle from the Java security providers list (only in case it was loaded by the Web Services Stack webapp classloader). In this case, you do not need to clean up the security providers or restart JRE.

Deploying Web Services Stack on an Apache Tomcat Installation

Before deploying Web Services Stack on an Apache Tomcat Installation, make sure the version of your Apache Tomcat Installation is the same as the version of Apache Tomcat used by the Software AG Common Platform.

To deploy Web Services Stack to your Apache Tomcat installation

1. Stop the Apache Tomcat Server.
2. Navigate to the <*CATALINA_HOME*>/conf/ directory.

3. Open the server.xml file.
4. Set the `unpackWars` parameter to `true` and save your changes.
5. Copy the wsstack.war file to the webapps directory of your Apache Tomcat installation.
6. Start Apache Tomcat.

   The content of the wsstack.war file are expanded into the wsstack directory under the webapps directory of your Apache Tomcat installation.

Managing Web Services

You can manage web services using the Axis 2 administration module. You can do the following:

- Upload service
- List available services and service groups
- List available modules and globally engaged Axis 2 modules
- List available phases
- View global chains and operation-specific chains
- Engage Axis 2 module for all services, for a service group, for a service, and for an operation
- Activate and deactivate services
- Edit service parameters

For more information on the Axis 2 administration module, see the Apache Tomcat documentation.

Accessing the Administration Module

Access the Web Services Stack administration module at http://*host* :*port* /wsstack/axis2-admin/

Changing Logon Credentials

By default, the administration module is secured by the administrator logon credentials configured in the axis2.xml file in the *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/conf directory. The default user name is admin and the default password is axis2. If you do not change the defaults, you may be exposed to a security threat through the administration module.

You can change the default user name with the *userName* parameter in the axis2.xml configuration file. To change the password, log on to the administration module and

click **Change Password** in the administration page header. If the Web Services Stack configuration file cannot be modified by the web application, you see the message `Password change is disabled`. In this case, you must use the Web Services Stack Reset Password Utility, below.

Changing the Administrator Password Using the Reset Password Utility

The Reset Password Utility is the resetPassword script stored in the *Software AG_directory*\WS-Stack\bin directory. The script requires write permission over the configuration file. After resetting the password, restart Web Services Stack for the changes to take effect.

Change the Administrator password as follows:

1. Retrieve the axis2.xml configuration file on the server.
2. Run the resetPassword script in the *Software AG_directory*\WS-Stack\bin directory.
3. Replace the original configuration file.
4. Restart Web Services Stack.

Displaying Deployed Web Services Stack Libraries

You can use the administration module provides to list deployed Web Services Stack libraries. The deployed libraries are JAR files that are installed with the Web Services Stack installation. You might use the list of these libraries for troubleshooting.

Go to http://*host* :*port* /wsstack/ in your browser. The default port for the deployment of Web Services Stack is 10010. Click the **Validate** link on the welcome page, then scroll down the Web Services Stack validation page.

5 Configuring the Java Service Wrapper

Determine Whether Your Product Uses the Java Service Wrapper, and Which Version

On the machine that hosts your Software AG products, open a command window and go to the Software AG installation directory. If you see a directory named profiles, one or more of your products uses the Java Service Wrapper. The names of directories within the profile directory correspond to profile names for the products (that is, *Software AG_directory*/profiles/*profile_name*). For example, the *Software AG_directory*/profiles/CTP directory is for the Software AG Runtime.

You will need to refer to the Tanuki Software, Ltd. website for detailed information about Java Service Wrapper properties listed in this guide. However, you will need to know which version of the Java Service Wrapper your product uses. To determine the version, go to the *Software AG_directory*/profiles/*profile_name* /bin directory and run the command `service -version`.

Editing Java Service Wrapper Properties

Each Software AG runtime product that runs on the Software AG Common Platform has two configuration files for the Java Service Wrapper.

- The wrapper.conf file contains the Java Service Wrapper property settings that are installed with the product. Never edit the contents of this file unless instructed to do so by Software AG.
- The custom_wrapper.conf file contains properties that override and modify the settings in the wrapper.conf file. If you want to edit property settings for a product's Java Service Wrapper, this is the file in which to do so.

> **Important:** Software AG products have different policies regarding the Java Service Wrapper properties you can configure. See the administrator's guide for your product before changing any Java Service Wrapper property settings.

To edit wrapper properties

1. Go to the *Software AG_directory* /profiles/*profile_name* directory for your product and open the wrapper.conf and custom_wrapper.conf files in a text editor.
2. Go to the Java Service Wrapper product documentation on the Tanuki Software, Ltd. website for detailed information about each property. Then go to the product documentation for any product-specific instructions.
3. If the property you want to edit already exists in the custom_wrapper.conf file, edit it in that file. If the property does not yet exist in the custom_wrapper.conf file, copy the property from the wrapper.conf file and then edit it in the custom_wrapper.conf file. If you are working with a sequenced attribute property, you must match the

sequence of properties in the custom_wrapper.conf file to the sequence of properties in the wrapper.conf file.

Important: Never edit the contents of the wrapper.conf file.

4. Save the custom_wrapper.conf file. Exit the wrapper.conf file without making any changes.
5. Restart the product.

Generating a Thread Dump Using the Java Service Wrapper Utility

A thread dump can help you locate thread contention issues that can cause thread blocks or deadlocks. The Java Service Wrapper provides a utility that enables you to generate thread dumps of the JVMs for Software AG products that are running as Window services.

On the machine that hosts your Software AG products, open a command window, go to the *Software AG_directory*/profiles/*profile_name* /bin directory, and run the command `service -dump`. The Java Service Wrapper writes the thread dump to the wrapper.log file in the *Software AG_directory*/profiles/*profile_name* /logs directory.

6 Using Command Central to Manage Software AG Runtime (CTP)

Configuration Types That OSGI-CTP-TOMCAT-ENGINE Supports

The OSGI-CTP-TOMCAT-ENGINE run-time component supports the following configuration types.

Configuration Type	Used to configure...
SIN-INTERNAL-GROUPS	The groups in the internal user stores.
COMMON-LOCAL-USERS	The internal users for Software AG Runtime.
COMMON-JAAS	The JAAS login modules to use for authentication and authorization, for example to allow authentication against external user stores.
COMMON-JVM-OPTIONS	Extended JVM options.
COMMON-JAVASYSPROPS	Java system properties.
COMMON-MEMORY	Common memory settings, such as Initial heap size and Maximum heap size.
COMMON-PORTS	The HTTP, HTTPS, JMX, JDWP (Debug), and SSH ports. By default, the HTTP, HTTPS, and JMX ports are enabled, and the JDWP and SSH ports are disabled.
COMMON-PROXY	The proxy server settings if you must route server requests through a third-party server.
SIN-INTERNAL-ROLES	The user roles in the internal user stores.

Lifecycle Actions for OSGI-CTP-TOMCAT-ENGINE

The following table lists the run-time statuses that the OSGI-CTP-TOMCAT-ENGINE run-time component can return in response to the `sagcc get monitoring state` command, along with the meaning of each run-time status.

Action	Description
Start	Starts the run-time component. When successful, the run-time status is set to ONLINE.
Restart	Stops, then restarts the run-time component. When successful, the run-time status is set to ONLINE.
Stop	Stops the run-time component. When successful, the run-time status is set to STOPPED. Stopping the component ends remote communications with the web user interface and the REST API.

Run-Time Monitoring Statuses for OSGI-CTP-TOMCAT-ENGINE

The following table lists the run-time statuses that the OSGI-CTP-TOMCAT-ENGINE run-time component can return in response to the `sagcc get monitoring state` command, along with the meaning of each run-time status.

Run-time Status	Meaning
ONLINE	The OSGI-CTP-TOMCAT-ENGINE run-time component is running.
STOPPED	The OSGI-CTP-TOMCAT-ENGINE run-time component is not running because it was shut down normally.
UNKNOWN	The status of the OSGI-CTP-TOMCAT-ENGINE run-time component cannot be determined.

7 Working with Software AGCommon Landscape Asset Registry

About Software AGCommon Landscape Asset Registry

Software AG Common Landscape Asset Registry (LAR) is a software library that provides components for managing user assets within a landscape of Software AG runtimes. LAR enables the process of repository-based deployment, in which runtimes pull assets from a repository. The process of deploying assets to the landscape is decoupled from the process of managing the number and type of runtimes.

LAR consists of two peer subsystems: a *registry* and a *repository*. The registry manages a set of assets stored in the repository. The registry contains a description of the assets and enables you to search for and filter assets by product.

The assets stored in the repository are binary artifacts and metadata. You can push assets to the repository, and then also pull those assets for deployment in a landscape.

You can use LAR with the JFrog Artifactory open-source software (OSS). You deploy the Artifactory to Command Central, and then configure LAR to work with the Artifactory.

Prerequisites for Using Common Landscape Asset Registry

You must have the following technologies and products installed:

1. Git server with Large File Storage (LFS) support.
2. Git client with LFS support. You must install Git LFS separately after installing the Git client. For more information about installing Git LFS, see the Git LFS documentation.
3. Software AG Command Central. For more information about installing Command Central, see *Software AG Command Central Help*.

Deploying the JFrog Artifactory to Command Central

Use the following procedure to deploy the JFrog Artifactory open-source software (OSS) as a web application to the OSGi profile of Command Central (CCE).

Before starting the deployment, you must stop Command Central, if it is running.

To deploy the Artifactory to the OSGi profile of Command Central

1. Download the latest version of the JFrog Artifactory OSS .zip file from "https://bintray.com/jfrog/artifactory/jfrog-artifactory-oss-zip/view".
2. Extract the contents of the file to a folder on your local file system.
3. Copy the artifactory.war file from *Artifactory_folder* /webapps to *Software AG_directory* /profiles/CCE/workspace/webapps.

4. Go to *Software AG_directory* /profiles/CCE/configuration and open the custom_wrapper.conf file in a text editor.
5. Add the following Java additional parameters, and then save the file:

```
set.ARTIFACTORY_HOME=%OSGI_CONFIGURATION_AREA%%WRAPPER_FILE_SEPARATOR%artifactory
wrapper.java.additional.100=-Dartifactory.home="%ARTIFACTORY_HOME%"
wrapper.java.additional.100.stripquotes=TRUE
wrapper.java.additional.101=-Dfile.encoding=UTF8
wrapper.java.additional.102=-Djruby.compile.invokedynamic=false
wrapper.java.additional.103=-Dartdist=zip
wrapper.java.additional.104=
 -Dorg.apache.tomcat.util.buf.UDecoder.ALLOW_ENCODED_SLASH=true
wrapper.java.additional.105=-XX:+UseG1GC
```

6. Create a folder named "artifactory" in the *Software AG_directory* /profiles/CCE/configuration directory.
7. Copy the *Artifactory_folder* /etc folder to the *Software AG_directory* /profiles/CCE/configuration/artifactory directory.
8. Go to *Software AG_directory* /profiles/CCE/configuration/artifactory/etc and open the binarystore.xml file in a text editor.
9. Add the following lines after the `<chain template="file-system"/>` element, and then save the file:

```
<provider id="file-system" type="file-system">
   <baseDataDir>Software AG_directory /profiles/CCE/workspace/data
   </baseDataDir>
</provider>
```

10. Verify that deployment is successful:
    a. Start Command Central.
    b. Go to http://*host* :*port* /artifactory, where *host* and *port* are the Command Central hostname and HTTP port.

    If deployment is successful, you should see the Artifactory home page. You can log into the Artifactory and create repositories.

Logging Into the JFrog Artifactory

Use the following procedure to log into the JFrog Artifactory deployed in the OSGi profile of Command Central (CCE).

To log into the Artifactory

1. Go to http://*host* :*port* /artifactory, where *host* and *port* are the Command Central hostname and HTTP port.
2. Log into the Artifactory, using the following credentials:
   - **Username** - admin
   - **Password** - password

Note: After you log into the Artifactory, you can change the default administrator password by going to **Edit profile > Current password > Change Password**.

Adding Repositories to the JFrog Artifactory

Use the following procedure to add binary repositories to the JFrog Artifactory deployed in the OSGi profile of Command Central (CCE). For more information about how to deploy the JFrog Artifactory to Command Central, see "Deploying the JFrog Artifactory to Command Central" on page 106.

To add a repository to the Artifactory

1. Log into the Artifactory as described in "Logging Into the JFrog Artifactory" on page 107.
2. Create a new repository:
   a. Go to http://*host* :*port* /artifactory/webapp/#/admin/repositories/local, where *host* and *port* are the Command Central hostname and HTTP port.
   b. Add a new repository and specify values for the fields, as required.

      Important: Select **maven-2-default** in the **Repository Layout** field to enable integration of the repository with the Common Landscape Asset Registry.

      For information about the fields and values to specify, see the JFrog Artifactory documentation.

You can upload artifacts to the newly added repository by using the Artifactory user interface or REST API. For more information, see the JFrog Artifactory documentation.

Configuring the Common Landscape Asset Registry to Use the JFrog Artifactory

Use the following procedure to configure Software AG Common Landscape Registry (LAR) to use the Artifactory deployed in the OSGi profile of Command Central (CCE).

To configure the Common Landscape Asset Registry

1. Create a properties file with the following name:

   com.softwareag.repository.maven.pid-*mavenRepoName* .properties

   where *mavenRepoName* is the name of your binary repository.
2. Add the following properties:

```
name=mavenRepoName
type=maven
```

```
remoteStore=http://host:port/artifactory
remoteStoreRepository=repoKey
username=username
@secure.password=apiKey
```

where:

- *mavenRepoName* is the name of your binary repository.
- *host* and *port* are the Command Central hostname and HTTP port number.
- *repoKey* is the repository key that you specify when you create a new repository.

  To obtain the repository key for an existing repository, in the Artifactory deployed to Command Central, go to the Admin panel and select **Repositories > Local**.
- *username* is the user name of an Artifactory user with administrator privileges.

  To add a new Artifactory user with administrator privileges, in the Artifactory deployed to Command Central, go to the Admin panel and select **Security > Users > New**. Specify values for the required fields and select the **Admin** checkbox.
- *apiKey* is the API key for the user.

  To obtain the API key for a user, in the Artifactory deployed to Command Central, go to **Edit Profile > Authentication Settings > API Key > Generate**.

For more information about the values to specify for the properties, see the Artifactory documentation.

3. Copy the properties file to *Software AG_directory* /profiles/CCE/configuration/com.softwareag.platform.config.propsloader.