Using Software AG Web Services Stack

Version 9.7

October 2014

Document ID: WEBM-WSS97-20141015

Table of Contents

About this Guide

This document provides information about using Web Services Stack.

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 (...).

Documentation Installation

You can download the product documentation using the Software AG Installer. The documentation is downloaded to a central directory named _documentation in the main installation directory (SoftwareAG by default).

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 and certified samples, 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, 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 Understanding Web Services Stack

Overview

Software AG Web Services Stack is a toolkit that enables users to create, configure, deploy, and manage web services. It handles the complex process of processing request and response messages between web services within Software AG products.

Web Services Stack enables you to create and expose services out of pure Java classes which implement business logic. You can create Web services which contain multiple Java classes essential for the completion of a particular business case.

When you create Web services, you can design them by providing individual configuration settings. 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.

Web Services Stack also enables you to deploy the Web services you have created, and expose their functionality to a broader usage. Depending on your environment, you can deploy the 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 various Web service clients.

Supported Web Services Standards

This section provides the list of web services standards supported by Software AG Web Services Stack.

- 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)

2 Configuring Web Services Stack

Software AG Web Services Stack provides a variety of configuration settings at repository, modules, and web services level.

Runtime Configuration

Available Configuration Files

Three kinds of global runtime configuration files exist:

- **axis2.xml** - used for configuring the client side and the server side of all deployed Web services. This file is located in the *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/conf directory.

  Caution: The axis2.xml file contains important information such as user name and password for the administration console login. System administrators must change the default credentials to protect the access to the axis2.xml configuration file.

- **services.xml** - used for configuring particular Web services. This file is located in the META-INF subdirectory within the service archive available in the *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/services directory.
- **module.xml** - used for configuring particular modules. This file is located in the META-INF subdirectory within the module archive available in the *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/modules directory.

Additional Web Services Stack specific configuration files include:

- **SMH Agent configuration files** - including the deployclient.properties file, used by Software AG System Management Hub for deploying web services, and the argusagent.properties file, containing information about the host name and server port of the deployed Web Services Stack. Both files are available in the *Software AG_directory*/WS-Stack/conf directory.
- **Client-side configuration files** - including the wsclientsec.properties file, containing security-related information. The file is available in the *Software AG_directory*/WS-Stack/bin directory. For more information about the security configuration, see "Message-Level Security" on page 22.

Available Web Services Stack Configuration Types

Web Services Stack Client Runtime Configuration

The Web Services Stack installation provides a client runtime configuration. It is primarily used by the Web Services Stack SMH agents, but it can also be used by any user-implemented Web services client. Web Services Stack is administered by an administration service deployed on it and is referred by the SMH agents.

The client runtime configuration uses the following files:

- Runtime configuration file: *Software AG_directory*/WS-Stack/repository/conf/axis2.xml.
- Repository directory: *Software AG_directory*/WS-Stack/repository, where
  - The Web services directory is *Software AG_directory*/WS-Stack/repository/services.
  - The modules directory is *Software AG_directory*/WS-Stack/repository/modules.

Web Services Stack Runtime Configuration in Standalone Server

Web Services Stack provides a Standalone Server that can be started without using a servlet container or an application server. When started using the *Software AG_directory*WS-Stack/bin/axis2server.bat/sh script, the Standalone Server uses the following runtime configuration files by default:

- Runtime configuration file: *Software AG_directory*/WS-Stack/conf/axis2.xml.
- Repository directory: *Software AG_directory*/WS-Stack/repository, where
  - The Web services directory is *Software AG_directory*/WS-Stack/repository/services.
  - The modules directory is *Software AG_directory*/WS-Stack/repository/modules.

Web Services Stack Runtime Configuration in Web Application

Web Services Stack is also distributed as a web application (wsstack.war file). In this case the repository is in the web application directory. You can find the wsstack.war file in the *Software AG_directory*/WS-Stack/webapp/wsstack installation directory. You can deploy this web application on any servlet container. The following runtime configuration is used:

- Runtime configuration file: *Software AG_directory*/WS-Stack/webapp/wsstack/WEB-INF/conf/axis2.xml.
- Repository directory: *Software AG_directory*/WS-Stack/webapp/wsstack/WEB-INF, where
  - The Web services directory is *Software AG_directory*/WS-Stack/webapp/wsstack/WEB-INF/services.
  - The modules directory is *Software AG_directory*/WS-Stack/webapp/wsstack/WEB-INF/modules.

Web Services Stack Runtime Configuration in Software AG Runtime

Web Services Stack is also integrated and distributed with Software AG Runtime server instance. Web Services Stack uses the following runtime configuration when the Software AG Runtime Server is started:

- Runtime configuration file: *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/conf/axis2.xml.

- Repository directory: *Software AG_directory*/profiles/CTP/workspace/wsstack/repository, where
  - The Web services directory is *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/services.
  - The modules directory is *Software AG_directory*/profiles/CTP/workspace/wsstack/repository/modules.

axis2.xml File Configuration

The axis2.xml is a configuration file provided by the Apache Software Foundation. For further information about the Axis2 parameters available in this file and their values, see the Axis2 Configuration Guide.

Setting up Additional Properties in axis2.xml

Web Services Stack uses some custom parameters in the axis2.xml configuration file. They are set on the server side of the Axis2 configuration. Web Services Stack uses the default value for these parameters. If you want to change the default value, you must add the parameter in the axis2.xml configuration file and provide a different value. This section contains additional information about the parameters and their values.

- `includeWrappedTypesDeclaration` - Defines whether to include message-wrapper elements in the WSDL XSD schema. Axis2 processes an RPC-style WSLD definition and automatically creates a wrapper element and type definition for each message. After that Axis2 processes internally any request or response as if it is in a document style with an element declaration for each message. Valid values are:
  - `true` (default) - Axis2 creates the web service instance and automatically adds the auto-generated types to the XSD of the original WSDL definition.
  - `false` - Axis2 creates a copy of the WSDL definition when processing the message types and modifies the copy instead of the original WSDL document.
- `enableWSDLValidation` - Defines whether to validate WSDL documents. Valid values are:
  - `true` - Axis2 validates the provided WSDL document against the external resources.
  - `false` (default) - Axis2 does not validate the WSDL document against the external resources.
- `enableSoapValidation` - Defines 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 are compliant 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 the following levels:
  - globally - you enable SOAP validation globally by setting the parameter in the axis2.xml file.
  - on service group level - you enable SOAP validation for a specific group by setting the parameter inside a ServiceGroup tag in the services.xml file.
  - on service level - you enable SOAP validation for a specific service by setting the parameter inside a Service tag in the services.xml file.
  - on operation level - you enable SOAP validation for a specific operation by setting the parameter inside an Operation tag in the services.xml file.
  - on message context level - you enable SOAP validation for a specific request by setting the parameter programatically to MessageContext.
- On the client side you can enable SOAP validation at the following levels:
  - globally - you enable the SOAP validation globally by setting the parameter in the axis2.xml file.
  - specific client execution - you enable the SOAP validation for operations that expect large SOAP messages by calling programatically `Options.setProperty("disableSoapValidation",Boolean.TRUE)`.

- `wsdl4jRegisterDefaultExtensionAttributeTypes` -Configures whether Axis2 registers default extension attribute types in the WSDL4J extension registry. Valid values are:
  - `True` - Axis2 registers default extension attribute types.
  - `False` (default) - Axis 2 does not register default extension attribute types.

  Configuration is done on Input, Output and Fault WSDL elements using String type.

Setting up Specific Message Builders and Message Formatters in axis2.xml

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 you will find some Web Services Stack specific information about the proprietary message builders and message formatters available in the axis2.xml configuration file.

Message Builders

The Web Services Stack axis2.xml file has defined proprietary message builders for the following content types to extend the default functionality provided by Axis2:

- text/xml
- application/xml

- application/soap+xml

The new 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>
```

Message Formatters

The Web Services Stack axis2.xml file has defined proprietary message formatters for the following content types to extend the default functionality provided by Axis2:

- text/xml
- application/xml
- application/soap+xml

The new 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>
```

Client-Side Configuration

Only one specific configuration for Web Services Stack exists on the client side.

securityConfigFile

The value of this parameter must be an absolute or a relative path either to the current working directory, or to the *repository path* /conf directory, and it must point to the wsclientsec.properties file containing security-related information. For example:

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

For more information about the security configuration, see "Message-Level Security" on page 22.

MTOM Configuration

Binary content often has to be re-encoded in order 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.

enableMTOM

Enables you to enforce MTOM message encoding. The parameter can be configured at any of the configuration levels (globally in the axis2.xml file, per service or per operation in the services.xml file), and it can have the following values:

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

3 Configuring Web Services Stack Security

Software AG Web Services Stack provides the following set of security features:

- Message content security
- HTTP basic authentication support
- Communication channel security using SSL
- User authentication based on Software AG Security Infrastructure Login Modules

Message-Level Security

Overview

Software AG 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.

Message-level security secures the message content itself, in contrast to transport-level security, where the communication channel is secured. To apply message security, you have to make several configurations on both the client side and the server side.

Web Services Stack provides an Software AG Designer plug-in graphical user interface that can be used to create the needed security configuration. You can install the plug-in in Eclipse and use it to create web service archives. For more information, see Software AG Designer Plug-in.

Security configurations in Web Services Stack are based on *WS-Security Policy specification*.

For more information about configuring message-level security according to your security needs, see Web Services Security: SOAP Message Security 1.1 and http://specs.xmlsoap.org/ws/2005/07/securitypolicy/ws-securitypolicy.pdf.

Prerequisites

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.

Server-Side Configuration

You can specify the following server-side configuration settings in the services.xml file:

For more information about setting up...	See...
Keystore caching	"Enabling Keystore Caching" on page 23
Password callback handler caching	"Enabling Password Callback Handler Caching" on page 23
`sp:RequiredElements` and `sp:RequiredParts` assertions validation	"Enabling sp:RequiredElements and sp:RequiredParts Assertions Validation" on page 26
WS-I Basic Security profile compliance mode	"Modifying the WS-I Basic Profile Compliance Mode" on page 26

Enabling Keystore Caching

Keystore caching can be enabled at the following levels:

- globally - in the axis2.xml file for all services that are deployed in Web Services Stack.
- per service, service group, or specific operation - in the services.xml descriptor file for your service.

Note: Since the keystore configuration can be different for each message, the caching is executed per message.

To enable keystore caching

1. Depending on the level at which you want to enable keystore caching, open the axis2.xml file or the services.xml descriptor file in an editor of your choice.
2. Set the `cacheCryptoInstances` parameter to `true`.

   ```
   <parameter name="cacheCryptoInstances">true</parameter>
   ```

3. Save your changes.

When a service is undeployed or simply stopped, any cached keystores will be removed. You can deactivate (stop) services using the "Using the WSS Administration Module" on page 77.

Enabling Password Callback Handler Caching

You can enable caching of initialized password callback handler classes to additionally improve performance. The callback handler instance is always cached on the service instance and will be lost once the service is undeployed.

To enable password callback caching

1. Open the services.xml file for editing.
2. Set the `cachePasswordCallbackHandler` parameter to `true`.

   ```
   <parameter name="cachePasswordCallbackHandler">true</parameter>
   ```

3. Save your changes.

Configuring Encryption User

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.

To configure the encryption user

1. Open the services.xml file for editing.
2. Set the `encryptionUser` parameter to `useReqSigCert`.

   ```
   <encryptionUser>useReqSigCert</encryptionUser>
   ```

   `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.

Symmetric Binding Security Configuration in the Services.xml File

The keystore properties can be configured by adding a Rampart custom policy assertion to the services.xml file. In the code sample below, the value `client` 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>client</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>
```

Enabling sp:RequiredElements and sp:RequiredParts Assertions Validation

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.

To enable `sp:RequiredElements` and `sp:RequiredParts` assertions validation

1. Open one of the following files for editing:
   - axis2.xml - to enable the change on the entire runtime.
   - services.xml - to enable the change on the particular Web service only.
2. Add the following parameters:

   ```
   <parameter name="enableRequiredElementsXPathCompatibility">true</parameter>
   <parameter name="enableRequiredPartsValidation">true</parameter>
   ```
3. Save your changes.

Alternatively, you can enable `sp:RequiredElements` and `sp:RequiredParts` assertions in the business logic of a Web service client using the following code snippet:

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

You can also make the changes described above using the Web Services Stack plug-ins available in Software AG Designer. For more information, see, *Software AG Designer Online Help*Web Services Security: SOAP Message Security 1.1, and WS-Security Policy Language.

Modifying the WS-I Basic Profile Compliance Mode

Web Services Stack enables you to enable or disable the WS-I Basic Profile compliance mode for your Web services.

For more information about the usage of the WS-I Basic Security Profile compliance mode, see WS-I Basic Profile.

To modify the WS-I Profile compliance mode

1. Open the services.xml file for editing.
2. Modify the value of `wsiBSPCompliant`.
   - Set to `true` (default value) to enable the WS-I Basic Profile compliance mode.
   - Set to `false` - to disable the WS-I Basic Profile compliance mode.

   ```
   <ramp:wsiBspCompliant>false</ramp:wsiBspCompliant>
   ```

3. Save the file.

Client-Side Configuration

When you use the client API to invoke web services that require security, you can specify security configuration settings through a properties file. For a list of supported configuration keys, see "Client-Side Security Configuration Parameters" on page 27.

To specify the properties file

1. Open the client axis2.xml for editing.
2. Set the `securityConfigFile` parameter to specify the file name and path to the custom properties file.

```
<parametername="securityConfigFile">D:/wsdev/SampleWSClient/wsclientsec.prop
erties</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, then the configuration loading routine does not throw any exceptions.

Note: The security configuration settings are loaded only if the web service policy contains security assertions.

Client-Side Security Configuration Parameters

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

Parameter and Description
USERNAME The user name used by: ■ The Web Services Stack UsernameToken function in the UsernameToken. ■ The 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. ■ The Web Services Stack encryption function if `ENCRYPTION_USER` is not set.
ENCRYPTION_USER The 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 value of the `USERNAME` parameter to get the certificate.

Parameter and Description
USER_CERTIFICATE_ALIAS The alias of the key pair in the keystore used for getting the private key for the signature. If this parameter is not set, the signature function uses the value of the `USERNAME` parameter.
STS_ALIAS The STS alias used as an encryption user in case of a STS authentication.
POLICY_VALIDATOR_CLASS The 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 written in the following simple date 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 A 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 A list of Xpath expressions which refer to nodes that must be MTOM-optimized. The configured value is a semicolon delimited list of Xpath expressions.

Parameter and Description
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
 	
A 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-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 The WSS4J-specific Crypto implementation that is to be used for generating the signature. It can be set to one of the following: ■ `org.apache.ws.security.components.crypto.Merlin` is the default one if the parameter is not set. ■ `org.apache.ws.security.components.crypto.BouncyCastle`
KEYSTORE_PROVIDER_SIGN The signature keystore provider. If not set the JVM uses the default keystore provider, usually Oracle. For more information, see the *java.security.Provider* javadocs.
KEYSTORE_TYPE_SIGN The signature keystore type. If not set, the JVM uses the default keystore type, usually JKS.

Parameter and Description
For more information, see the *java.security.KeyStore#getDefaultType() method* javadocs.
KEYSTORE_FILE_SIGN The signature keystore file.
KEYSTORE_PASSWORD_SIGN The signature keystore password.
CRYPTO_PROVIDER_ENCRYPT The WSS4J-specific Crypto implementation to use for encryption. It can be set to one of the following: ■ `org.apache.ws.security.components.crypto.Merlin` - this is the default one if the parameter is not set. ■ `org.apache.ws.security.components.crypto.BouncyCastle`
KEYSTORE_PROVIDER_ENCRYPT The encryption keystore provider. If not set the JVM uses the default keystore provider, usually Oracle. For additional information, refer to the *java.security.Provider* javadocs.
KEYSTORE_TYPE_ENCRYPT The encryption keystore type. If not set, the JVM uses the default keystore type, usually JKS. For additional information, refer to the *java.security.Provider* javadocs.
KEYSTORE_FILE_ENCRYPT The encryption keystore file.
KEYSTORE_PASSWORD_ENCRYPT The encryption keystore password.
CRYPTO_PROVIDER_STS The WSS4J-specific Crypto implementation to use for protection in case of a STS authentication. It can be set to one of the following:

Parameter and Description
■ `org.apache.ws.security.components.crypto.Merlin` is the default one if the parameter is not set. ■ `org.apache.ws.security.components.crypto.BouncyCastle`
KEYSTORE_PROVIDER_STS The keystore provider used in case of a STS authentication. If not set, the JVM uses the default keystore provider, usually Oracle. For additional information, refer to the *java.security.Provider* javadocs.
KEYSTORE_TYPE_STS The keystore type used in case of a STS authentication. If not set, the JVM uses the default keystore type, usually JKS. For additional information, refer to the *java.security.KeyStore#getDefaultType() method* javadocs.
KEYSTORE_FILE_STS The keystore file used in case of a STS authentication.
KEYSTORE_PASSWORD_STS The keystore password used in case of a STS authentication.
SSL_KEYSTORE_TYPE The type of the keystore specified by the `KEYSTORE_SSL_LOCATION` parameter.
SSL_KEYSTORE_PASSWORD The password for the keystore specified by the `KEYSTORE_SSL_LOCATION`. This parameter corresponds to the JSSE `javax.net.ssl.keyStorePassword` system property.
KEYSTORE_SSL_LOCATION The keystore file for SSL authentication. This parameter corresponds to the JSSE `javax.net.ssl.keyStore` system property. For more information, see the JSSE Reference Guide. **Note:** Specifying the keystore is required only if the remote SSL server requires client authentication.

Parameter and Description
TRUSTSTORE_SSL_LOCATION The truststore file for SSL authentication. The server certificate must be installed in this truststore and it must be trusted. This parameter corresponds to the JSSE `javax.net.ssl.trustStore` system property. If the property is not set, the client uses the <*java-home*>lib/security/jssecacerts and <*java-home*>/lib/security/cacerts in that order. For more information, see the JSSE Reference Guide.
TRUSTSTORE_SSL_PASSWORD The password for the truststore specified by the `TRUSTSTORE_SSL_LOCATION`. This parameter corresponds to the `javax.net.ssl.trustStorePassword` system property. For more information, refer to the JSSE Reference Guide.

Note: The last five parameters refer to transport-level security configuration (SSL settings).

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.

Transport-Level Security

Prerequisites for the Setup and Use of Transport-Level Security

Web Services Stack enables you to set up transport-level security configuration, where you secure the communication channel instead of the message data itself. The most typical case of transport-level security is the use of HTTP transport over SSL.

This section provides details on the configuration and usage of web service communication over HTTPS.

Configuring Software AG Runtimeto Use SSL at the Server Side

You can set up Software AG Runtime to explicitly use the HTTPS transport for web service communication.

Important: If you define the use of HTTPS transport in the services.xml file, you must not define a transport listener in the 2.xml file. Software AG Runtime will automatically register the transport listener for you based on the HTTPS connector.

To configure Software AG Runtimeto use SSL at the server side

1. Navigate to <*Software AG_directory*>/profiles/CTP/configuration/com.softwareag.platform.config.propsloader and open the com.softwareag.catalina.connector.https.pid-<*PORT*> .properties file to configure an SSL Connector.
2. Define the following properties:

Property name	Description
clientAuth	Defines whether a certificate must be required from the client or not. Set to ■ `true` - to require a valid certificate chain from the client before accepting a connection. ■ `want` - to request a client certificate chain, but not fail if one is not presented. ■ `false` - (default) to not require a certificate chain.
sslProtocol	The version of the SSL protocol to use. The default value is `TLS`.
keystoreFile	The path to the keystore file where you have stored the server certificate to be used to decrypt the requests and encrypt the responses.
port	The TCP port number on which this Connector will create a server socket and await incoming connections. The default value is `8443`.
keystorePass	The password that provides access to the certificate from the keystore file.

Property name	Description
scheme	The configured scheme necessary for the SSL communication. Set the value to `https`.
enableLookups	`false`
secure	`true`
maxSpareThreads	`75`
maxThreads	The maximum number of request processing threads to be created.
keystoreType	The type of keystore file to be used for the server certificate.
disableUploadTimeout	Allows the use of a different, longer connection timeout. The attribute is set to `true` if not specified.
algorithm	The certificate encoding algorithm to be used.
minSpareThreads	The number of request processing threads that will be created when this connector is first started.
acceptCount	The maximum queue length for incoming connection requests when all possible request processing threads are in use. The default value is `100`.
maxHttpHeaderSize	The maximum size of the request and response HTTP header, specified in bytes. If not specified, this attribute is set to `4096` (4 KB).
keyAlias	The alias that identifies the key pair in the keystore. If not specified, the first key found in the keystore will be used.

Following is a sample code listing for an SSL connector configuration:

```
clientAuth=false
sslProtocol=TLS
SSLEnabled=true
keystoreFile=<KEYSTORE_FILE_PATH>
enabled=trueport=10011
keystorePass=<KEYSTORE_PASSWORD>
keyAlias=<ENCRYPTION_KEY_ALIAS>
scheme=https
enableLookups=false
```

```
secure=true
alias=defaultHttps
maxSpareThreads=75
maxThreads=150
server=SoftwareAG Runtime
keystoreType=JKS
disableUploadTimeout=true
description=Default HTTPS Connector
algorithm=SunX509
minSpareThreads=25
acceptCount=100
maxHttpHeaderSize=8192
```

Note: The default value of the connector port is `10011`. The port number is calculated during the installation of the product.

Configuring SSL at the Client Side

To configure SSL at the client side

1. The client must send a request against HTTPS endpoint with a port that is equal to the one specified at server side (in previous example `10011`).
2. Set the properties in your security configuration file. You can configure this file as a parameter in the 2.xml configuration file:

   ```
   <parametername="securityConfigFile">your client security config file
   path</parameter>
   ```

   For more information on the 2.xml configuration file, see "axis2.xml File Configuration" on page 16.

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

   Or:

   Use the Web Services Stack client API to set the required properties:

   ```
   //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 following security properties at the client side relate to the SSL configuration:

Property name	Description
SSL_KEYSTORE_TYPE	The type of the keystore specified under the KEYSTORE_SSL_LOCATION.
SSL_KEYSTORE_PASSWORD	The password for the keystore specified under KEYSTORE_SSL_LOCATION. This property corresponds to the `JSSE`

Property name	Description
	`javax.net.ssl.keyStorePassword` system property.
KEYSTORE_SSL_LOCATION	The keystore file for SSL authentication. This property corresponds to the `JSSE javax.net.ssl.keyStore` system property. Note that specifying the keystore is required only if the remote SSL server requires client authentication. For more information, refer to the *JSSE Reference* Guide.
TRUSTSTORE_SSL_LOCATION	The truststore file 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 falls back to <java-home>lib/security/jssecacerts and <java-home>/lib/security/cacerts in that order. For more information, refer to the *JSSE Reference* Guide.
TRUSTSTORE_SSL_PASSWORD	The password for the truststore specified under TRUSTSTORE_SSL_LOCATION. This property corresponds to the `javax.net.ssl.trustStorePassword` system property. For more information, refer to the *JSSE Reference* Guide.

SSL with Client Authentication

Server-Side Configuration

Software AG Web Server based on Apache Tomcat may also be configured to use a client certificate to encrypt the transferred data.

Using Client Authentication with Software AG Web Server

To use client authentication with Software AG Web Server

1. Navigate to *Software AG_directory*/profiles/CTP/configuration/com.softwareag.platform.config.propsloader and open the com.softwareag.catalina.connector.https.pid-<*PORT*> .properties
2. Set `clientAuth` to "`true`".
3. Set the keystore properties.
4. Set the truststore properties.

Important: You can also configure the truststore location of Software AG Runtime by starting it with the respective Java system property, because if the truststore properties are not set in your configuration, Software AG Web Server uses the default Java trusted authority keystore.

Configuring the Truststore Location of Software AG Runtime by Using the Respective Java System Property

To configure the truststore location of Software AG Runtimeby starting it with the respective Java system property

Add the following options when starting in *Software AG_directory*/profiles/CTP/configuration/config.ini:

```
javax.net.ssl.trustStore=your_path_to/truststore.jks
javax.net.ssl.trustStorePassword=your_password)
```

Use the following settings to configure the truststore properties in the HTTPS connector:

Property name	Description
truststoreFile	The TrustStore file to use to validate client certificates.
truststorePass	The password to access the truststore. This defaults to the value of `keystorePass`.
truststoreType	Add this element if your are using a different format for the truststore than you are using for the keystore. The `keystoreType` defaults to "JKS".

Following is a sample of the connector configuration:

```
clientAuth=true
sslProtocol=TLS
SSLEnabled=true
keystoreFile=<KEYSTORE_FILE_PATH>
truststoreFile=<TRUSTSTORE_FILE_PATH>
truststorePass=<TRUSTSTORE_PASSWORD>
```

```
truststoreType=<TRUSTSTORE_TYPE>
enabled=true
port=10011
keystorePass=<KEYSTORE_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
```

Note: If you encounter a problem with a service that declares the usage of HTTPS, see "Prerequisites for the Setup and Use of Transport-Level Security" on page 32.

Client-Side Configuration

It is also possible to use client certificate with the Web Services Stack client, although additional work is needed to use the Java 1.4 compatible HTTP sender (utilizing the Jakarta Commons `HttpClient` component). In order to make the `Commons HttpClient` use client certificate for the encryption one needs to register a new HTTPS socket factory since the default one does not handle the case with the client certificate. The `Commons HttpClient` library does not provide the appropriate socket factory implementation but there is one in the contrib package (commons-httpclient-contib) that is part of the commons-httpclient project, namely `AuthSSLProtocolSocketFactory`. This can be set in the following way:

```
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);
```

Setup and Use of HTTP Basic Authentication

The basic HTTP authentication is a common transport security mechanism. The server sends requests to the client to provide its credentials in an HTTP authorization header. Thus the mechanism provides both authentication and authorization means. 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 (rampart). More information and examples of how to set up your basic HTTP authentication scenario is given below.

Validating Basic HTTP Authentication in Rampart

To validate basic HTTP authentication in 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. For more information, see *Client Authentication*.

To avoid malfunction of the functionality, the `wsstack` must be running inside a servlet container or a server such as Software AG 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 Web service Security Policy. The following security policy snippet 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 of an `sp:HttpsToken` assertion which means that the server also requires the usage of HTTPS transport. To use this feature you must engage rampart for your Web service. To engage rampart for the Web Service, add the following lines in the service descriptor (services.xml):

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

Add a policy that contains the`sp:HttpBasicAuthentication` to your Web service (such as in the example snippet above).

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

Configuring your Web Service Client to Use HTTP Basic Authentication

To configure your Web service client to use HTTP basic authentication

1. Supply the `HttpTransportProperties.Authenticator` object.
2. Set the user name to "`wssuser`".
3. Set the password to "`wssuser`".
4. Set this configuration as an option of the web service client.

Following is a sample code listing of a web service client implementation when you want to use 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.2.transport.http.HTTPConstants.AUTHENTICATE,auth);
```

Important: In the example above, you must supply the `HttpTransportProperties Authenticator` object first, and then set up a user name and a password. Finally, you need to set this configuration as an option of the web service's client.

Client Authentication

Overview

Software AG Web Services Stack provides a mechanism for authenticating clients in Web Services Stack runtime layer using the common Java Authentication and Authorization Service (JAAS) security framework.

Software AG Security Infrastructure (SIN) provides you with JAAS-based `LoginModules` for client authentication.

When you log on using JAAS `LoginContext, a javax.security.auth.Subject` is produced. That subject contains `user 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 SIN `SagCredentials`. After that, the logon process is performed in the policy validator implementation of Rampart.

JAAS Configuration

Before you can log on, you must configure JAAS. For information about the JAAS configuration file, see SIN documentation in the *webMethods Product Suite* directory on the Software AG Documentation website.

Security Credentials

There are two types of user credentials that are used for authentication in Web Services Stack:

Transport-level credentials

Transport-level credentials refer to the communication channel used for the message exchange and are specific for the respective transport that is used. Web Services Stack extracts those credentials from the HTTP(S) transport only:

- User name and password, in the case of a basic HTTP authentication.
- A client certificate chain in the case of a client certificate used for encryption of the transferred data.

Message-level credentials

In the case of message-level credentials, Web Services Stack can extract those from the SOAP security header:

- A user name and a password if you use `UsernameToken` with plain text password.
- X509Certificate used for the signatures if there are signed parts or elements in the message.

Implementation of Password Callback Handlers

User implemented password callback handlers are used to:

- Retrieve passwords to be placed inside a UsernameToken (corresponding to a given user name).
- Retrieve passwords to access user private keys from a keystore (the keystore password itself is directly set in the ramart configuration)
- Verify password in received UsernameToken (corresponding to a given user name).

The callback handlers can retrieve passwords from configuration files, data bases, LDAP servers, or other application components which are used for user management (for example Security Infrastructure).

Web Services Stack has a predefined set of password callback handlers, which facilitate different scenarios for retrieving passwords. You can use these handlers directly or you

can develop your own password callback handlers out of them. The following password callback handlers are available:

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 configuration file of the Web Services Stack (axis2.xml). You can provide a configuration file to the callback handler as follows:

- You can specify the configuration file 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 as follows:

- You can specify 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_properties_file_location
  </parameter>... </service></serviceGroup>
  ```

  - 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) 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 users.xml 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) 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 users.xml and is searched on the class path of the client. Then it is loaded as a resource.

Implementations of Policy Validation Callbacks

In the wsstack-jaas.jar module, there are ready-to-use policy validator implementations that may be configured and used easily to log on.

Following are examples of those implementations:

- `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` (the default value is WSS_Transport_IS) and for message-level credentials logging on

under `sin.jaas.msg.login.context` (the default value is `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.

Note: To use one of the preceding callbacks, specify the `policyValidatorCbClass` in the Rampart policy assertion.

Authentication Steps

To authenticate using SIN

You must include the path to SIN JAR in the classpath (in Software AG Runtime, SIN artifacts are already available for Web services deployed as "aar" archives). All classes that are used in the JAAS configuration file must also be set in the classpath.

1. Configure the JAAS configuration file.
2. Configure a web service to do the following:
   - Specify the policyValidatorCbClass in the Rampart configuration policy assertion.

     Following is a sample code listing of 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:policyValidatorCb
     Class>com.softwareag.wsstack.jaas.callback.MultiLoginPolicyValidatorCall
     back
     </ramp:policyValidatorCbClass>
     ```

   - Specify the `LoginContext` name as a parameter on one of the web service levels (global level in axis2.xml; service group level in the 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>
     ```

With those settings, you are authenticated when logging on by Security Infrastructure.

For information about the authentication steps, see SIN documentation in the *webMethods Product Suite* directory on the Software AG Documentation website.

4 Configuring Web Services Stack Transports

Software AG Web Services Stack supports sending and receiving of messages over the following transports:

- HTTP or HTTPS
- TCP
- JMS
- Mail

The following instructions show how to configure and activate or deactivate the transports supported by Web Services Stack.

HTTP and HTTPS Transport

Overview

By default, the HTTP transport is activated in all three Web Services Stack distributions (Web Services Stack in a standalone server, Web Services Stack in a web application, and Web Services Stack in Software AG Runtime. The HTTPS transport is activated only in Web Services Stack in web application distribution, and needs to be explicitly enabled in the other installations. The following procedure outlines the steps you must perform to enable or disable the HTTP and HTTPS transports.

Note: If you disable a connector in Software AG Runtime, make sure that the respective transport sender and receiver are disabled on Web Services Stack side as well (axis2.xml). If a transport is enabled in the axis2.xml file and the respective connector in the server.xml file is disabled, an error occurs in Web Services Stack.

Note: Make sure at least one of your HTTPS connectors has the `description` property set to `Default HTTPS Connector`. This value exists by default in the predefined HTTPS connector definition. The `Default HTTPS Connector` value is used by Software AG Common Platform to distinguish default connectors from other existing connectors. If this value is not present in at least one HTTPS connector definition, this results in an invalid or corrupted Software AG Runtime configuration the next time you install or upgrade a product in the profile.

Activating or Deactivating HTTP or HTTPS Transport

Activating or Deactivating HTTP or HTTPS in Web Services Stack

To activate or deactivate HTTP or HTTPS in Web Services Stack

1. Go to the web application server where Web Services Stack is installed.
2. Open the axis2.xml configuration file under the webapps/wsstack/WEB-INF/conf directory for the Web Services Stack web application.

> **Note:** For Web Services Stack that uses Software AG Runtime, the axis2.xml file is in the following directory: <*Software AG_directory*>/profiles/CTP/workspace/wsstack/repository/conf/.

3. 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" ... />
```

4. Restart Web Services Stack for the modifications to take effect.

   For Web Services Stack that uses Software AG Runtime, restart the Software AG Runtime Windows Service (using the Control Panel > Administrative Tools > Services).

Activating or Deactivating HTTP or HTTPS in Software AG Runtime

To activate or deactivate HTTP or HTTPS in Software AG Runtime

1. Navigate to the <*Software AG_directory*>/profiles/CTP/configuration/com.softwareag.platform.config.propsloader/ directory.
2. Open the file that defines the connector to be activated or deactivated, for example, com.softwareag.catalina.connector.http.pid-<*identifier* >.properties)
3. To enable or disable the connector, in the properties file edit the value of the `enabled` property to true or false respectively.

> **Note:** When you save the properties file, the change will be automatically detected and Software AG Runtime will update itself - no restart is required.

TCP Transport

Activating TCP Transport (Server-Side Configuration

To activate TCP transport in Software AG Web Services Stack

1. Go to the web application server where Web Services Stack is installed.
2. Open the axis2.xml configuration file under the webapps/wsstack/WEB-INF/conf directory for the Web Services Stack web application.

> **Note:** For Web Services Stack that uses Software AG Runtime, the axis2.xml file is in the following directory: <*Software AG_directory*>/profiles/CTP/workspace/wsstack/repository/conf/.

3. 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 default value is `6060`.

Note: Restart Web Services Stack for the modifications to take effect.

Enabling WS-Addressing

Since the TCP transport has no application level headers (and no target endpoint URI), you need WS-Addressing to dispatch the service.

Note: WS-Addressing may not be enabled in the default Web Services Stack installation.

To enable WS-Addressing

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

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

- 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>
  ```

- Enable WS-Addressing by using the Web Services Stack Designer plug-in. To do so, select **Enable WS-Addressing** from the **Modules** list in the **Services** tab.For more information about working with the Web Services StackSoftware AG Designer plug-in, see *Software AG Designer Plug-in*.

Forcing Deployment Over TCP Transport Only

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.

To deploy over TCP transport only

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

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

```
</service>
```

- 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 (Client-Side Configuration)

To make a call to a web service over TCP transport, configure the client's repository.

To invoke a web service over TCP

1. 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" ... />
   ```

2. Engage globally the WS-Addressing module (`addressing.mar`) in the client's axis2.xml file:

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

3. Ensure the WS-Addressing module (`addressing.mar`) is present in the /modules directory in the client's repository

JMS Transport in Web Services Stack Web Application

Prerequisites

Following are guidelines to the prerequisites for the activation of JMS transport.

Installing and Starting a Message Broker

To install and start a message broker

In order to achieve JMS communication, you need a message broker that handles the distribution of messages between communicating parties. Web Services Stack does not include a built-in message broker. This requires the use of an external one. Apache ActiveMQ is an open source message broker that you can download from *http://activemq.apache.org/activemq-411-release.html*.

Extract the files from the downloaded archive into a directory of your choice. For example, ACTIVEMQ_HOME.

After the installation, ActiveMQ is running with a basic configuration that is sufficient for its integration with Web Services Stack.

Note: If you want to terminate the broker, type the CTRL-C command in the command prompt in which it is running.

Running the ActiveMQ Message Broker

To run the ActiveMQ message broker

1. Open the command prompt.
2. Navigate to the ACTIVEMQ_HOME/bin directory.
3. Run the activemq.bat file.

You can find more about installing and using Apache ActiveMQ open source message broker at *http://activemq.apache.org/getting-started.html*.

Providing Additional Libraries for Web Services Stack

Configuring Web Services Stack to work with Apache ActiveMQ message broker requires the provision of additional libraries for Web Services Stack.

To provide additional libraries for Web Services Stack

1. Go to the ACTIVEMQ_HOME/lib directory.
2. Copy the following libraries to the <web_app_server>/webapps/wsstack/WEB-INF/lib directory of Web Services Stack in the web application server:
   - activemq-core-4.1.1.jar
   - activeio-core-3.0.0-incubator.jar
   - geronimo-jms_1.1_spec-1.0.jar
   - geronimo-j2ee-management_1.0_spec-1.0.jar

   Note: You need these libraries for any client that invokes a service over JMS transport.

Activating JMS Transport (Server-Side Configuration)

To activate JMS transport in Web Services Stack

1. Go to the web application server, where Web Services Stack is installed.
2. Open the axis2.xml configuration file under the webapps/wsstack/WEB-INF/conf directory for the Web Services Stack web application.

Note: For Web Services Stack that uses Software AG Runtime the axis2.xml file is in the following directory: <*Software AG_directory*>/profiles/CTP/workspace/wsstack/repository/conf/.

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

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

4. 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 default for use by services that do not explicitly specify in their services.xml configuration file the connection factory they want to use.

   Those connection factories are associated with Apache ActiveMQ implementation whose libraries are required for Web Services Stack. Each connection factory specifies the following parameters:

   - An initial naming factory class
   - Naming provider URL
   - The JNDI name of an actual JMS connection factory.

   Web Services Stack can run with the default configuration of Apache ActiveMQ. 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.

Forcing 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.

Note: 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.

Deploying over JMS Transport Only

To deploy over JMS transport only

Configure the web service's services.xml file by adding the element in bold:

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

Or:

Use Web Services StackDesigner plug-in 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 directly the services.xml file, or by using the Web Services StackDesigner plug-in. The parameters that define the connection factory name are optional. If they are not specified, the service uses the default connection factory (named as `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 elements in bold. The connection factory can be any of the connection factories defined in axis2.xml and the destination name can be anything. "transport.jms.ConnectionFactory" and "myQueueConnectionFactory" are samples for values of parameters.

```
<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>
```

You can also use the Web Services StackDesigner plug-in to specify the connection factory name.

To specify the connection factory name using the Web Services StackDesigner plug-in

1. In the **Project Explorer** view, select the Web service archive that will use the connection factory.
2. Click the **Services** tab.
3. To specify the connection factory, in the **Properties** section:

   a. Click **Add**.
   b. Type "transport.jms.ConnectionFactory" in the **Name** field.
   c. Type "myQueueConnectionFactory" (or another connection factory defined in axis2.xml) in the **Value** field.
   d. Click **OK**.
4. To add the destinations:
   a. Click **Add**.
   b. Type "`transport.jms.Destination`" in the **Name** field.
   c. Type "`dynamicQueues/TestQueue`" (or other value of your choice) in the **Value** field.
   d. Click **OK**.

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

For more information about working with the Web Services StackDesigner plug-in, see *Software AG Designer Plug-in*.

Invoking a Web Service Over JMS Transport (Client-side Configuration)

To make a call to a web service over JMS transport, you have to configure the client's repository.

To invoke a web service over JMS

1. 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" ... />
```

2. Engage globally the WS-Addressing module (addressing.mar) in the client's axis2.xml file.

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

3. Ensure the WS-Addressing module (addressing.mar) is present in the /modules directory in the client's repository.

Mail Transport in Web Services Stack Web Application

Prerequisites

The activation of mail transport in Web Services Stack requires a mail server that transfers e-mail messages. The Apache Java Enterprise Mail Server (James) is an open source SMTP and POP3 mail server that is used by Web Services Stack.

After you have installed and configured your mail server, you have to create accounts. You need to create a mail account that represents the e-mail address of Web Services Stack. Additional accounts can be created to correspond to different clients.

Installing Apache James Server

To install Apache James server

1. Download the archive with the binary distribution of the Apache James mail server from the Apache James website.
2. Extract the files from the downloaded archive to a JAMES_HOME directory of your choice.
3. Start and stop the mail server once so that it unpacks its configuration files.

Opening the Configuration Files for Editing

To open the configuration files for editing

1. Open the command prompt.
2. Navigate to JAMES_HOME/bin directory.
3. Run run.bat to start the server.
4. Use the CTRL+C command to stop the mail server.
5. Type the `ipconfig/all` command to check your network configuration.

Note: You need this information for the next instruction (configuring the DNS servers).

Configuring the DNS Servers in the Mail Server

To configure the DNS servers in the mail server

1. Open the config.xml file under the JAMES_HOME/apps/james/SAR-INF directory.
2. 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>
```

   Note: Apache James mail server requires the valid IP addresses of the DNS servers in your network configuration.

3. Start the mail server again.

For more information on configuring the Apache James mail server, see the Apache James documentation.

Creating Accounts in the Mail Server

To create an account

1. Start the Apache James mail server if it is not started.

   To start Apache James Server, run the console command prompt, navigate to JAMES_HOME/bin directory and run `run.bat`.

2. Start James Remote Manager Service (this tool is used for administration purposes).

   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.

3. 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 ID and the password, unless you have changed them.

   Type "`root`" for the logon ID and for the password.

4. Create the account. The command for adding a new user is `adduser username password`. After executing the command, you get a confirmation.

   Type the following command:

   ```
   adduser server wsstack
   ```

5. 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 4555JAMES Remote Administration Tool 2.3.1
   Please enter your login and passwordLogin id:rootPassword:root
   Welcome root. HELP for a list of commandadduser server wsstack
   User server addedquitBye
   ```

Activating Mail Transport

There are prerequisites for the activation of mail transport. Refer to the following instructions and the description of the required parameters for the transport receiver and the transport sender.

To activate mail transport in Web Services Stack

1. Open the axis2.xml configuration file under the webapps/wsstack/WEB-INF/conf directory for the Web Services Stack web application.

Note: For Web Services Stack that uses Software AG Runtime the axis2.xml file is in the following directory: <*Software AG_directory*>/profiles/CTP/workspace/wsstack/repository/conf/.

2. Configure the context root of Web Services Stack.

   In the axis2.xml file, find the parameter with the name `contextRoot`. Uncomment it (if it is commented) and ensure that its value is "`wsstack`":

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

3. Activate the mail transport receiver and the mail transport sender.

   In the axis2.xml file find and 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 fake default values. They need to be verified.

Required Parameters for the Transport Receiver

The following table lists the required parameters and their description:

Parameter	Description
mail.pop3.host	The host name (or IP address) where the James mail server is running. The host name (or IP address) where the James mail server is running. If the server is running on the same machine as Web Services Stack, then the value can be "`localhost`" or "`127.0.0.1`".
mail.pop3.user	The user name of a user registered in the James mail server. The user name in the following sample code is the user registration from the example in the preceding topic "Creating Accounts in the Mail Server" on page 57.
transport.mail.pop3.password	The user's corresponding password for his account.
mail.store.protocol	The value "`pop3`" is expected for that parameter.

Parameter	Description
transport.mail.replyToAddress	This parameter is responsible for the following values: ■ 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 James mail server, separated by the @ sign. **Note:** 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`".
transport.listener.interval	Controls the time interval (in milliseconds) for checking the mail server for new messages. **Note:** This parameter is optional. If omitted, its default value is = "`3000` " milliseconds (which equals 3 seconds).

Following is a sample code listing of 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>
```

Required Parameters for the Transport Sender

The following table lists the required parameters and its description:

Parameter	Description
mail.smtp.host	The host name (or IP address), where James mail server is running. It corresponds to the `mail.pop3.host` parameter under the Mail transport receiver.

Parameter	Description
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.

Following is a sample code listing of the usage of the required parameter 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>
```

Forcing Deployment Over Mail Transport Only

If you want to restrict a web service to be deployed only over Mail transport, you must add the following element in the web service's services.xml file:

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

If not configured explicitly, a web service is deployed over all activated transports in Web Services Stack.

Invoking a Web Service Over Mail Transport

To call a web service over mail transport, configure the client's repository.

To configure the client's repository

1. In the client's axis2.xml configuration file, find and uncomment the sections that define the transport receiver and transport sender with `name="mailto"`:
2. 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.

   For more information, see *Activating Mail Transport*.

Sample Client Configuration

Following is a sample code listing of 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>
```

5 Configuring Web Services Stack Monitoring and Logging

This document covers the logging facility and the utility for monitoring of SOAP messages.

SOAP Monitor in Web Services Stack

Overview

The distribution of Software AG Web Services Stack comes with a SOAP monitor that allows users to monitor SOAP messages exchanged between Web service clients and Web services running in Web Services Stack.

SOAP messages are shown with the structure that 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 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, take a binary data shown into a TCPMon (general purpose tcp monitor). To make easy to understand, only 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 that a SOAP monitor shows is the following:

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

As you can see, the binary data is shown “by value”. This is because it was already processed by the system phases of the Axis 2 engine.

Using SOAP Monitor in Web Services Stack Web Application

SOAP monitor is disabled by default.

To enable SOAP monitor

1. Open the web.xml file that is located in the WEB-INF directory of the wsstack webapp.
2. Uncomment the `<servlet-name>SOAPMonitorService</servlet-name>` part.
3. Uncomment the `<servlet-mapping>` part.
4. Copy the following `SOAPMonitor` classes from soapmonitor folder and paste them directly under the expanded `wsstack` context root:

   `org\apache\axis2\soapmonitor\applet\SOAPMonitorApplet$ServiceFilterPanel.cl`

```
assorg\apache\axis2\soapmonitor\applet\SOAPMonitorApplet$SOAPMonitorData.cl
assorg\apache\axis2\soapmonitor\applet\SOAPMonitorApplet$SOAPMonitorFilter.
classorg\apache\axis2\soapmonitor\applet\SOAPMonitorApplet$SOAPMonitorPage.
classorg\apache\axis2\soapmonitor\applet\SOAPMonitorApplet$SOAPMonitorTable
Model.classorg\apache\axis2\soapmonitor\applet\SOAPMonitorApplet$SOAPMonito
rTextArea.classorg\apache\axis2\soapmonitor\applet\SOAPMonitorApplet.class
```

Important: Ensure you keep the classes packaging structure.

5. Engage the `soapmonitor` Axis 2 module globally in the axis2.xml by adding the following line:

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

   You can engage it in the same way for a service in the services.xml file.

6. Restart the servlet container or the `wsstack webapp`.
7. Go to http://<*host* >:<*port* >/wsstack/SOAPMonitor to start using the SOAP monitor.

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

Using SOAP Monitor in Web Services Stack on Software AG Runtime

SOAP monitor is disabled by default.

To enable SOAP monitor

1. Open the axis2.xml file that is located in the <*Software AG_directory*>\profiles\CTP \workspace\wsstack\repository\conf directory.
2. Engage the `soapmonitor` Axis2 module globally in the axis2.xml by adding the following line:

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

   You can engage it the same way for a service in the services.xml file.

3. Add a soapMonitorPort parameter which defines the port to use for communication with the SOAP Monitor Applet

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

   Note: If the port parameter is missing or empty, the SOAP Monitor servlet will not be available.

4. Restart Software AG Runtime.
5. Go to `http://<host>:<port>/wsstack/SOAPMonitor` to start using the SOAP monitor.

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

Logging in Web Services Stack Web Application

Overview

Software AG Web Services Stack uses Apache Commons Logging (JCL) and its `log4J` facility. The JCL provides thin-wrapper log implementations for other logging tools, including the default `log4J`.

For more information on `log4J` and logging services, see the Apache documentation.

Note: The distribution of Web Services Stack comes with a log4j.properties file and a commons-logging.properties file by default. You can find them in <*Web Services Stack_directory*>/webapp/wsstack/WEB-INF/classes.

Note: Those files are also included in the wsstack.war web archive in <*Web Services Stack_directory*>/webapp, in case you deploy Web Services Stack another servlet container or application server.

To enable `log4J` place the commons-logging.properties file into the given module classpath.

Log4J Logging Levels

The log4j.properties files come with a default value of the logging level. You can change those values according to the requirements of your system.

The default logging level is `info`. Following are the standard levels in descending (in terms of priority) order:

- `fatal`
- `error`
- `warn`
- `info`
- `debug`
- `trace`

Note: A lower level covers all levels above it. For example, if warn is set, then all logs of level error and fatal are logged, too.

It is important to ensure that the log messages are appropriate in content and severity. See the following table for guidelines on the usage of logging levels:

Logging Level	Usage
`fatal`	Severe errors that cause premature termination. Expect these to be immediately visible on a status console.
`error`	Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.
`warn`	Use of deprecated APIs, poor use of API, error-like situations, other runtime situations that are undesirable or unexpected, but not necessarily incorrect. Expect these to be immediately visible on a status console.
`info`	Interesting runtime events (start /shut down). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.
`debug`	Detailed information on the flow through the system. Expect these to be written to logs only.
`trace`	More detailed information. Expect these to be written to logs only.

Logging in Web Services Stack on Software AG Runtime

Overview

Software AG Web Services Stack running on Software AG Runtime uses Journal Logging as a logging mechanism. The Journal Logging is delivered with the following shared component bundle: `com.softwareag.sc.core` and its configuration file is located in the following directory on the file system: <*Software AG_directory*>/profiles/<profile>/configuration/logging/log_config.xml

Basically, the format of the log_config.xml file is the same as the format of the Log4J XML configuration. Apart from that, the Journal Logger contains several additional appenders than the standard Log4J appenders.

To enable logging and configure the corresponding severity, edit the log_config.xml file and edit the following excerpt as shown in the sample below:

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

Log4J Integration and Logging Levels

The Journal Logger is a wrapper around Log4J and every Journal Logging (JL) logger wraps a standard Log4J logger. For this reason the JL component delivers Log4J as part of its implementation. The JL configuration is a standard Log4J configuration that sets up the underlying Log4J library. If you need you can use Log4J directly. You should add your Log4J settings to the JL configuration file.

The default logging level is `info`. Following are the standard levels in descending order (in terms of priority):

- `fatal`
- `error`
- `warn`
- `info`
- `debug`
- `trace`

Note: A lower level covers all levels above it. For example, if warn is set, then all logs of level error and fatal are logged, as well.

It is important to ensure that the log messages are appropriate in content and severity. See the following table for guidelines on the usage of logging levels:

Logging Level	Usage
`fatal`	Severe errors that cause premature termination. Expect these to be immediately visible on a status console.
`error`	Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.
`warn`	Use of deprecated APIs, poor use of API, error-like situations, other runtime situations that are undesirable or unexpected, but not necessarily incorrect. Expect these to be immediately visible on a status console.
`info`	Interesting runtime events (start /shut down). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.
`debug`	Detailed information on the flow through the system. Expect these to be written to logs only.

Logging Level	Usage
`trace`	More detailed information. Expect these to be written to logs only.

System Management Hub Agents Logging

Software AG Web Services Stack provides a logging mechanism for its agent programs that use the Software AG System Management Hub administration functionality. These agent programs are called System Management Hub agents. They manipulate the Web Services Stack environment under the System Management Hub web interface.

If you experience problems when using the administration tool, you must enable the logging for the System Management Hub agents to see a detailed message log. It is recommended to use this logging mechanism only when you want to search for faults in the operation of the system. Otherwise, the performance of your interface may decrease.

To enable logging on System Management Hub agents

1. Start the web interface of System Management Hub in a web browser.
2. To open the registry editor, navigate to:

   Managed Hosts > <*host_name*> > System Management Hub > Registry > HKEY_LOCAL_MACHINE\SOFTWARE\Software AG > System Management Hub > Products > Web Services Stack 9.6 > Versions > 9.6 > Parameters.
3. Right click the **Parameters** node.
4. On the menu that opens, click **Modify Value**.
5. Switch the value of the registry parameter `enableLog` to "`1`".
6. Click **OK**.

You can find the output log file in <*Software AG_directory*>/WS-Stack/argus/wsstack.log.

6 Administering the Deployment of Web Services Stack

Software AG Web Services Stack contains a web application that can be deployed into any servlet container or application server that supports the Java Servlet standard. The deployable web archive is called wsstack.war and comes with the installation.

The option for generation, configuration and deployment of web services onWeb Services Stack allows the products to wrap an existing (legacy) server application and make it accessible to clients. Generally speaking, the product services are exposed as web services and deployed to a running Web Services Stack application.

Overview

The admin servlet is accessible at the following URL:

`http://<host_name>:<port_number>/wsstack/`

Note: Note:The default Software AG Runtime port is 10010. In case of deployment in custom application servers, the port is configured by the corresponding server administration tools.

After the deployment of the deployable web archive, you can click one of the links:

- Services

  See a list of the available deployed services. After installation, you have deployed only the Version service (that gives information for the version of Axis 2) and Web Services Stack.

- Validate

  Check whether the required libraries for the system are in place and view system information

- Administration

  Access the administration tool installation.

Bouncy Castle JCE Provider

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. Once 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

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.

Deploying Web Services Stack on JBoss 4

To deploy Web Services Stack to JBoss 4

1. Stop JBoss.
2. Create a wsstack.war directory in the *JBoss_directory*\server\default\deploy directory.
3. Unzip the content of the wsstack.war file into the wsstack.war directory.
4. Start JBoss.

Deploying Web Services Stack on JBoss 5

To deploy Web Services Stack on JBoss 5

1. Stop JBoss.
2. Create a wsstack.war directory in the *JBoss_directory*\server\default\deploy directory.

3. Unzip the content of the wsstack.war file into the wsstack.war directory.
4. Create a jboss-classloading.xml file with the following content:

```
<classloading xmlns="urn:jboss:classloading:1.0"
  name="wsstack.war"
  domain="wsstack"
  export-all="NON_EMPTY"
  import-all="true">
</classloading>
```

5. Copy the created jboss-classloading.xml file in the *JBoss_directory* /server/deploy/wsstack.war/WEB-INF directory.
6. Start JBoss.

Deploying Web Services Stack on BEA WebLogic 10.3

The Web Services Stack application can be used on BEA WebLogic server versions 10.3.0 to 10.3.5.

To deploy Web Services Stack on BEA WebLogic

1. Stop the BEA WebLogic server.
2. Copy the wsstack directory from *Software AG_directory*\WS-Stack\webapp\wsstack directory into the autodeploy directory.
3. Start the BEA WebLogic Server.

Deploying Web Services Stack on WebSphere 7.0.0.19

To deploy Web Services Stack on WebSphere 7.0.0.19

1. Copy the *Software AG_directory*/WS-Stack/webapp/wsstack.war file to a temporary location.
2. Unzip the war file.
3. Copy all the .mar files from WEB-INF/modules to WEB-INF/lib and change their extensions to .jar.

   Note: There might be an issue with mapping of MAR files when using Microsoft Office. When you have Microsoft Office installed, you cannot rename those files using Windows Explorer. In this case, use the `copy` command prompt command. For example, `<TEMP_Directory>\WEB-INF\modules>copy addressing-1.41.mar addressing-1.41.jar` copies the MAR file and changes its extension from MAR to JAR.

4. Recreate the WAR file using an application which supports creation of .zip files.

5. Log on to the Administrative Console and navigate to **Applications > New Application > New Enterprise Application**.
6. Enter the location of the wsstack.war file or browse for it using the **Browse** button, and then click **Next**.
7. Select the **Fast Path - Prompt only when additional information is required** radio button, and then click **Next**.
8. Click **Next** and leave the default values for the options in the **Step 1 Select installation options**, **Step 2 Map modules for servers**, and **Step 3 Map virtual hosts for Web modules** screens.
9. Click **Next** to go to the **Step 4 Map context roots for Web modules** screen, and type `wsstack` in the **Context root** field.
10. Click **Save** to save the changes to the master configuration.
11. Navigate to **Applications > Application Types > Web Sphere Enterprise Applications**.
12. Select **wsstack_war** to open the configuration dialog.
13. In the configuration dialog, click the **Class loading and update detection** link.
14. For **Class loader order**, select the **Classes loaded with local class leader first (parent last)** radio button.
15. For **WAR class leader policy**, select the **Single class leader for application** radio button.
16. Navigate to **Applications > Application Types > WebSphere Enterprise Applications**.
17. Start the Web Services Stack web application.

7 Using the WSS Administration Module

You can manage Web services using the administration module.

Note: The administration functionality is also available through System Management Hub (SMH). For details, see the System Management Hub documentation.

The information on the administration module is organized under the following sections:

- Accessing the Administration Module
- The Administration Module Features for Managing Web Services
- Displaying Deployed Libraries
- Changing Login Credentials
- Resetting the Password Utility in WSS Web Application

Accessing the Administration Module

Use the following URL to access the Web Services Stack administration module:

`http://<host_name>:<port_number>/wsstack/axis2-admin/`

For example: `http://localhost:10010/wsstack/axis2-admin/`.

The administration module is secured by default with administrator's logon credentials configured in the axis2.xml configuration file.

- If Web Services Stack is deployed as a web application, the file location is WEB-INF/conf/axis2.xml.
- If Web Services Stack is deployed on Software AG Runtime, the file location is <*Software AG_directory*>/profiles/CTP/workspace/wsstack/repository/conf/axis2.xml.

To log on to the administration module

1. Provide the logon credentials that are specified in the axis2.xml configuration file. The default user name is "admin" and the default password is "axis2".

   See Changing Login Credentials for details on changing the default user name and password at first login.

The Administration Module Features for Managing Web Services

Following are the features of the administration module for managing Web services:

- Upload service
- List available services

- List available service groups
- List available modules
- List globally engaged modules
- List available phases
- View global chains
- View operation specific chains
- Engage module for all services
- Engage module for a Service Group
- Engage module for a Service
- Engage module for an Operation
- Deactivate service
- Activate service
- Edit parameters of a service

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

Displaying Deployed Libraries

This section shows you how to display a list of the deployed Web Services Stack libraries.

The administration module provides you with an easy access to the list of the deployed libraries with information about them such as library name, JAR file details, and version number. The deployed libraries are JAR files that are installed with the Web Services Stack installation or at the deploy time of its web archive. You might use the list of these libraries for troubleshooting.

To display a list of the deployed Web Services Stack libraries

1. Type `http://<host_name>:<port_number>/wsstack/` in your browser.

   Note: The default port for the deployment of Web Services Stack on Software AG Runtime is `10010`.

2. Click the **Validate** link on the welcome page.
3. Scroll down the Web Services Stack validation page.

Changing Login Credentials

With the Administration Tool, the wsstack argus agents that perform all the administration tasks use the security settings provided by the product. In this case, your web services are secured.

With the administration module, there are default user credentials for logging on to it. If you do not change them after Web Services Stack is installed, you may be exposed to a security threat through the administration module.

Note: In case you want to connect to Web Services Stack from System Management Hub, provide the logon credentials for the administration module.

Web Services Stack provides you with the option to change the user credentials for the administration module. The user name can be changed in the configuration file using a text editor. The password, however, must not be modified by editing the axis2.xml file. Use the graphical user interface of Web Services Stack to change the password.

Changing the User Name

To change the user name

1. You can change the default user name with the *userName* parameter in the axis2.xml configuration file.

Changing the Password

To change the password

1. Log on to the administration module.
2. Click on the **Change Password** button in the administration page header.

   Note: If the Web Services Stack configuration file cannot be modified by the web application, you are notified that the password change is disabled with the "Password change is disabled" error. In this case, you must use the Reset Password Utility of Web Services Stack.

Resetting the Password Utility in WSS Web Application

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

This utility is used in the following scenarios:

- Restoring a password that you have forgotten

  Because the password is transformed into a hash, using an algorithm that cannot be reversed, it is not possible to restore a forgotten password. In that case, you can reset it using the Reset Password Utility.

- Changing the default password when you have received the "Password change is disabled" error.

  Following is a list of the cases in which you can receive this error:

  - Web Services Stack web application WAR is archived upon deployment
  - Web Services Stack web application WAR is not archived upon deployment, but the web application is not granted write permissions for the WEB-INF/conf/axis2.xml file.
  - Web Services Stack web application has not been deployed with the standard configuration file (WEB-INF/conf/axis2.xml). Instead, a URL configuration file or a JAR resource configuration file has been used.

Resetting a Forgotten Password

To reset a forgotten password

1. Run the resetPassword script in the *Software AG_directory*\WS-Stack\bin directory.
2. Restart Web Services Stack for the changes to take effect.

Changing the Password when WSS Web Application WAR is Archived upon Deployment

To change the password when the Web Services Stack web application WAR is archived upon deployment

1. Retrieve the configuration file.
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 for the changes to take effect.

Changing the Password when WSS Web Application WAR is not Archived upon Deployment

Use the following procedure if you want to change the password when the Web Services Stack web application WAR is not archived upon deployment, but the web application is not granted write permissions for the WEB-INF/conf/axis2.xml file.

To change the password when the Web Services Stack web application WAR is not archived upon deployment

1. Retrieve the configuration file.
2. Run the resetPassword script in the /bin directory.
3. Replace the original configuration file.
4. Restart Web Services Stack for the changes to take effect.

Changing the Password when WSS Web Application is not Deployed with the Standard Configuration File

Use the following procedure if you want to change the password when the Web Services Stack web application has not been deployed with the standard configuration file (WEB-INF/conf/axis2.xml).

To change the password when the Web Services Stack web application has not been deployed with the standard configuration file

1. Retrieve the configuration file.
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 for the changes to take effect.