TIBCO iWay® Service Manager

Security Guide

Version 8.0 and Higher
March 2021
DN3502115.0321

Contents

Chapter 1

Introducing Security

A secure system is a system on which enough trust can be put to use it together with sensitive information. To reach this trust, the system needs confidentiality, integrity, and authentication.

- Confidentiality ensures that information is not accessed by unauthorized persons.
- Integrity ensures the data has not been tampered with.
- Authentication ensures that users are who they claim to be.

This chapter provides a background on computer security, which sets the foundation for the following chapters that describe how these concepts relate to iWay Service Manager.

In this chapter:

Confidentiality

Confidentiality is achieved through encryption. A plain text is scrambled using a key into an unintelligible cipher text. The process is inverted with decryption where the cipher text is transformed into the original plain text. This procedure is called symmetric encryption because the same key is used for encryption and decryption. The key must be known to both parties ahead of time. For a particular cryptographic algorithm, the longer the key the more secure it will be. The difficulty to transmit the key is the main disadvantage of symmetric encryption.

In asymmetric encryption, a key pair is made up of the public key and the private key. The two keys are related mathematically, but it is not possible to deduce one from the other. The public key can be freely published whereas the private key must be kept a secret by the owner of the key. Indeed, asymmetric encryption offers no protection if the private key is compromised. The public and private keys are inverses of each other. Encrypting with the public key produces a cipher text that can only be decrypted by the private key. This can be done by everyone to ensure the message can only be understood by owner of the private key. Inversely, encrypting with the private key produces a cipher text that can be decrypted by the public key. Since everyone can have the public key, this cipher text is not secret, but it proves that it originated from the owner of the private key.

Public key encryption imposes considerable computational overhead and is not appropriate for securely transmitting large amounts of data. It is more feasible to use public key encryption to send a symmetric key, which can then be used to encrypt additional data. This is the approach used by the SSL protocol.

Integrity

Tampering detection relies on a mathematical function called a one-way hash. A hash is a function that reduces a variable amount of data to a fixed-size quantity. The hash is computed by the originator and sent together with the data. The recipient recomputes the hash from the data and compares against the hash it received. The data has not been modified if the two values match. The hash is not sufficient to prove data integrity because an attacker can replace both the message and the hash in transit. To address that, we need digital signatures.

Authentication

Most server software permits client authentication by means of a user ID and password. For example, a server might require a name and password before granting access to the server. The server maintains a list of names and passwords to which it will grant access.

Another form of authentication is a client certificate with a digital signature.

Public key cryptography employs certificates to avoid impersonation. A certificate is a binding between a subject identity and a public key. In other words, a certificate is a document that claims the embedded public key belongs to that subject. Anyone can produce a certificate, so how can we have confidence the claim is true? The answer is to delegate the verification.

- **Certificate Authority (CA).** Certificate Authority (CA) is an entity that follows certain procedures to verify the public key really belongs to the subject and the subject is really who he claims to be. Obtaining a certificate involves a one-time exchange between the CA and the subject. The subject creates a *Certificate Request* which also creates a new key pair. The subject keeps the private key secret and sends the certificate request (which contains the public key) to the Certificate Authority. The CA performs all the checks including validating the requester. The CA responds with a *certificate request response*, which is used to update the requester's key pair.

  A CA publishes its certificate to allow applications to verify the signature in the certificates it issued. A *Root CA* is a CA that signed its own certificate. This is common for commercial companies like VeriSign, or for internal projects that know they can trust themselves. A CA that publishes a certificate signed by another CA is called an *Intermediate CA*.

- **Certificate Chain.** Certificate Chain is an ordered sequence starting with the subject certificate up to the Root CA where each certificate is followed by the certificate that signed it. A self-signed certificate has a chain of depth 1. Most certificates have a chain of depth 2 consisting of the subject certificate and the Root CA. A certificate chain of depth 3 including an Intermediate CA is less common.

  A CA may revoke a certificate it issued if it determines the certificate is no longer valid before its expiration date. For example, this can happen if an employee leaves the company or the private key is compromised. The *Certificate Revocation List* contains the list of all certificates the CA revoked.

- **Keystore.** A Keystore is an object that holds keys. It can contain symmetric keys, private keys or public keys in the form of certificates. A keystore is usually a binary file, but it can also be implemented as a crypto module in hardware.

- **Certstore.** A Certstore is an object that holds certificates and Certificate Revocation Lists. This is different than a keystore because there may not be any provisions to store symmetric or private keys. In iWay Service Manager, a Certstore can be implemented as a keystore, a collection of files in a directory, or with LDAP.

- **Truststore.** A Truststore is a Certstore that contains exclusively the signing certificate of Trusted CAs.

- **Digital Signature.** A Digital Signature is a binding between a document and an identity. This is achieved by encrypting the hash of the document with the private key of the signer. The signature is then sent together with the document. The recipient needs to follow these steps to verify the signature. It recomputes the hash over the original document. It finds the certificate of the originator to obtain its public key. It decrypts the signature using the public key to obtain the hash computed by the originator. The signature is valid if the two hash values match. This proves the document has not been tampered with after it was signed. Furthermore, we know the document came from that originator because only the originator knows the private key used in the signature. The identity in the originator certificate can be trusted if the certificate has been signed by one of the Trusted CAs.

  Certificates can be used to replace user name/password schemes. In SSL, the server can ask the client to authenticate itself. During the handshake the server asks the client to send the client certificate together with a signed piece of random data. The server can ascertain the identity of the client by verifying that signature.

Chapter 2

Security in iWay Service Manager

This section describes the areas concerned with security within iWay Service Manager.

In this chapter:

Introducing Security Components

iWay Service Manager (iSM) is designed to service secure message traffic. Security within the iSM framework focuses on the following areas:

- Message acquisition and disposition using standards-based message protocols such as SSL and AS/2.
- Identification and rejection of spurious messages including DOS-type attacks.
- Standards-based cryptography to process secure messages such as S/MIME messages.
- XML Digital Signature for non-repudiation of messages.
- Authentication and authorization for connection and access to secure services within the server.

- Policy-based protected configuration, which prevents tampering or unauthorized changes to system components and their use during message processing.
- Masking of stored passwords.
- Security related functions in the iWay Functional Language.

This manual discusses each of these areas in terms of the security mechanisms being applied and the ways in which these mechanisms fit into the use of the server.

Note: iWay realizes that hackers and attackers will constantly create new and ingenious means to circumvent security. iWay will consider it a bug if such attacks result in the compromise of a secure protocol or security mechanism and will address such a defect on an expedited basis upon the presentation of necessary reproduction or diagnostic data.

Areas of Responsibility

No system can be secure unless good security practices are employed by the developers of the system and by those charged with running the system on a daily basis. iWay Service Manager uses advanced security mechanisms to acquire and emit protected messages, and to work with those messages during their transit through the server. iWay assumes good security practices by those running the server, including:

- Generation of private keys and acquisition/distribution of public keys and certificates.

  Note: iWay recommends the use of secure utilities for this purpose. As standards vary between users and countries, iWay does not offer these tools except for testing and demonstration purposes.
- Protection of configuration passwords.
- Physical protection of the software itself from detailed inspection and modification.
- Management of secure files, such as key stores and trust stores by the appropriate software.

  Note: iWay recommends the use of secure utilities for this purpose. As standards vary between users and countries, iWay does not offer these tools except for testing and demonstration purposes.
- Employment of good security practices including proper testing and validation of the software.

Users should protect their private keys. In particular, this means physical access to the machine must be protected. Should a private key become compromised, others could potentially gain access to sensitive information or forge digital signatures.

Message Acquisition and Disposition

Message acquisition and disposition refers to the secure receipt and distribution (emit) of messages. Messages on any channel can also be encrypted and decrypted using services provided by iWay Service Manager. Specific security protocols that are provided with iWay Service Manager are listed and described in the following table:

Protocol	Description
nHTTP	HTTP including SSL and support for both basic and digest authentication.
nAS2	AS2 secure messaging protocol.
FTP[/S]	FTP client over secure SSL channels.
SFTP	FTP client over secure shell.
FTPServer	FTP server offering FTP/S protocol.
MQ [with crypto]	Access to standard MQ security settings.

For more information on these message protocols, see the *iWay Service Manager Component and Functional Language Reference Guide*. All of these protocols are designed to automatically encrypt and decrypt messages, apply authentication and authorization accordingly, and attempt to isolate and prevent spurious message attacks.

Rejection of Spurious Messages

Invalid messages must be rejected to avoid denial of service attacks. Spurious message identification includes detection of invalid protocols, buffer overflow detection, large message rejection, IP address spoofing and the filtering of peer addresses through white lists of acceptable sources of messages.

Secure Multi-Purpose Internet Mail Extension (S/MIME)

S/MIME provides a way to send secure MIME data through encryption, digital signatures and compression. Many email programs support S/MIME to send secure messages. S/MIME can also be used in other protocols. For example, AS2 builds on S/MIME to provide a secure protocol on top of HTTP.

iWay Service Manager components related to S/MIME are listed and described in the following table:

Component	Description
nAS2	AS2 secure messaging protocol.
XDSMIMEPackerAgent	A service to create S/MIME messages.
XDSMIMEUnpackerAgent	A service to unwrap and verify S/MIME messages.
XDSmimePreParser	A preparser to unwrap and verify S/MIME messages.
XDNewSMIMEPE	A preemitter to serialize the document and attachments to S/MIME.

For more information on these components, see the *iWay Service Manager Component and Functional Language Reference Guide*.

XML Digital Signature

XML Digital Signature components support the creation and verification of Digital Signatures over an XML document and its attachments. It is possible to add reference key material to identify the signing key either directly in the Signature or in a BinarySecurityToken.

iWay Service Manager components related to XML Digital Signatures are listed and described in the following table:

Component	Description
XDInsertSAMLAssertionAgent	Inserts a SAML assertion in a new SecurityTokenReference element.
XDInsertWSSETimestampAgent	Inserts a WSSE timestamp in an XML document.
XDInsertWSSETokenAgent	Inserts a WSSE SecurityToken in an XML document.

Component	Description
XDXMLDSigCreateAgent	Creates an XML Digital Signature.
XDXMLDSigVerifyAgent	Verifies an XML Digital Signature.
XDXAdESCreateAgent	Creates an XML Advanced Electronic Signature.
XDXAdESVerifyAgent	Verifies an XML Advanced Electronic Signature.

For more information on these components, see *Security Services* on page 71.

Authentication and Authorization

Authentication is the process by which an application verifies the identity of users. Authorization is the process that determines the permissions of an authenticated user within the application. In iWay Service Manager (iSM), these closely related functions are performed using security realms. A security realm is a database of user information that can be used to validate the credentials of a user. Realms also include the list of security roles that have been assigned to each user. Thus, authorization is a matter of checking to see whether a user has the role required to access a resource or perform an action.

iSM listeners implementing protocols with standard authentication mechanisms, such as NHTTP, NAS2, and TelnetD, are tightly integrated with security realms. However, authentication and authorization can be added to any channel using the Authenticate/Impersonate service and the _hasrole() iFL function. For more information, see *Configuring Runtime Security Using Access Control* on page 35.

Using Policies

Security within the server is controlled using policies that can be configured. Policies can only be set by an operator with administrative privileges. Most policies can be set on a general (installation) or a configuration basis.

There are two basic kinds of policies:

- **Boolean.** A boolean policy is either on or off.
- **Value.** A value policy is set to a specific value to be applied to the server.

Policies currently in effect can be displayed by the *show policies* command from a shell terminal. To set a policy, use the following command:

```
set policy <[configname/]name> <value> [-noverify]
```

The security-based policies available in the server are listed in the following table:

Name	Default	Use
minpswd	0	Minimum number of characters needed for a password to operate the configuration consoles.
signpflow	false	Determines whether process flows must be signed for message execution.
signdict	false	Determines whether the catalog for the configuration/dictionary must be signed for the server to start.
signconfig	false	Determines whether the configuration file, which defines the configurations, must be signed for a server to start.

Although configuration files are generally signed, the policies control whether the signature is checked before the configuration file is used.

Policies are contained in a security file in the main (<iwayhome>/config) area of the server. This file is always signed and verified on each use, which prevents unauthorized access to the policies as a result.

Password Masking

All passwords identified as such to iWay Service Manager (iSM) are masked when stored in iSM configuration files. The masking is designed to prevent unauthorized disclosure of the password. The algorithm used is a salted value cipher that produces values suitable for storage in XML files. Password policies such as minimum password length apply only to iSM passwords, and not to passwords stored by iSM for use in other systems. For example, the password associated with iSM as a client of an external FTP server is controlled by the policies of that server; iSM simply masks the password during its storage.

Security Related iFL Functions

iWay Service Manager provides a scripting language, called iFL (iWay Functional Language), which is documented in the *iWay Service Manager Programmer's Guide* and *iWay Service Manager Component and Functional Language Reference Guide*.

iFL can be used to configure server components and to manipulate information. Several iFL functions are offered to assist in constructing secure applications. For example, hash functions are often used to add validation hashes to database records or to messages. While the details of these functions are discussed elsewhere, some of the most useful functions are listed and described in the following table.

Function	Description
_aes()	Encrypt or decrypt data using the AES algorithm.
_encr()	Masks the value using a storage masking algorithm that is provided by the server.
_getprin()	Return the user name or password from the current principal.
_hasrole()	Determines whether the authenticated source of this message has the authority to perform some operation.
_sha1()	Generates a hash over a series of values using the SHA1 algorithm.
_sha256()	Generates a hash over a series of values using the SHA256 algorithm.
_md5()	Generates a hash over a series of values using the MD5 algorithm.

Restricted XPath Expressions

Some security service parameters accept an XPath expression to locate an element in the XML document. These parameters support the full syntax of the XPath engine that is selected by the XPath Version parameter. For convenience, the security services can construct the path to the element if it is not found in the XML document. This is useful to ensure that a parent element exists before a new XML child is inserted at that location. Unfortunately, XPath is ambiguous when describing the construction of a path. For example, the element name might be missing (`/root/*`), some parents might be unknown (`//ds:Signature`), or siblings might be unknown (`/root/child[3]`).

To allow the creation of the path, you must restrict the syntax to a subset of XPath 1.0. A Restricted XPath expression has the following form:

```
/step1/step2/...
```

A step has the following form:

```
ns:elem[predicate]
```

The predicate is optional and has the following form when it is present:

```
[@ns1:attr1='val1' and @ns2:attr2='val2' and ...]
```

The namespace prefixes are optional, but when present, they must be declared in the XML Namespace provider. To evaluate a step:

1. Look under the current node to find the first element named *elem*, in the namespace declared with the prefix *ns*, which has all the specified attributes equal to the given value.
2. If there is a match, make the child element the current node.
3. If there is no match, create a new child element with all the attributes assigned to the specified value and make the new child the current node.

Because Restricted XPath creates the missing nodes as it progresses, there is no backtracking to find alternative paths. Notice that the document root element cannot be constructed by Restricted XPath. This is to respect the XML limitation of a single root element in an XML document.

As a special case, a pair of consecutive steps can have the following form:

```
*[1]/self::ns:elem[predicate]
```

Similar to XPath 1.0, this indicates that the requested element must be the first child of its parent. If the first child does not match, then create a new child element as mentioned in step 3 and insert it first in the list of children.

The Create Parent Element parameter tells the service whether to construct the path leading to the parent element, and consequently tells the service which XPath syntax to accept. When the parameter is set to *false*, the full expressive power of the XPath engine is available. However, the parent element must already exist in the XML document, otherwise an exception is generated. When the parameter is set to *true*, the path to the element will be constructed if necessary, but the expression syntax must be Restricted XPath.

The following Restricted XPath expression can create a SOAP Header before an existing SOAP Body:

```
/soapenv:Envelope/*[1]/self::soapenv:Header/wsse:Security
```

It is sometimes desirable to create multiple WSSE Security Headers in the same document because they are destined for different actors. The services must choose the WSSE Security Header with the correct actor when inserting the new content. The solution is to write the XPATH expression with a predicate for the actor attribute. The following is an expression that selects the mandatory WSSE Security Header destined for the next role even if there are other WSSE Security headers present:

```
/soapenv:Envelope/*[1]/self::soapenv:Header/
wsse:Security[soapenv:mustUnderstand='1' and
soapenv:actor='http://schemas.xmlsoap.org/soap/actor/next']
```

This expression is valid for full XPATH 1.0 or Restricted XPATH. When the Create Parent Element parameter is set to *true* and the WSSE Security Header is not found, then Restricted XPath creates a new WSSE Security Header and its two SOAP attributes.

Chapter 3

Security Providers

This section describes the available security providers that can be configured for iWay Service Manager.

An iWay Service Manager provider is an object that can be configured separately and then referred to by name in the configuration of other components. This helps maintenance by centralizing the configuration of commonly used properties. It also helps conserve memory because the underlying resource the provider represents can be created once and shared among all components that refer to the provider. It is possible to create multiple providers of the same type.

The providers that relate specifically to security are the Keystore, the SSL Context, the Directory CertStore, and the LDAP providers.

In this chapter:

Keystore Provider

The Keystore provider holds the necessary configuration to access a Keystore. A Keystore is an object that can hold symmetric keys, private keys or public keys in the form of certificates. A Keystore is usually a binary file. It can also be implemented as a crypto module in hardware and accessed through Sun's PKCS11 provider. See *nCipher Configuration* on page 203 for an example using the nCipher crypto module.

Keystore providers are usually referred to by name in other components, but you can also declare a default Keystore provider for SSL, and a (possibly different) Keystore provider for SMIME.

The Reload Period property tells the provider how often to check whether the Keystore should be reloaded. By default, the Keystore is loaded only once the first time the provider is accessed and will never be reloaded. When defined, the Reload Period is the minimum time to wait before a reload can occur. The check occurs only when the provider is accessed so there is no cost if there is no activity. The value 0 means the provider must check for a possible reload every time it is accessed. This is not as expensive as it may appear since the Keystore file will be reloaded only if the file was modified since the last check as determined by the file time stamp.

The Keystore provider is usually configured with the required password. The provider can also request the password at runtime if the application security requirements forbid storing the Keystore password in the configuration. This is done using a user-provided callback that queries for the password the first time the Keystore provider is accessed. This mechanism might be desirable for more secure Keystores implemented in hardware, such as an nCipher device.

The following class is a sample callback implementation. This callback can be enabled by setting the Callback Handler property to com.example.SampleCallbackHandler.

```
package com.example;
import javax.security.auth.callback.CallbackHandler;
public class SampleCallbackHandler implements CallbackHandler
{
      public SampleCallbackHandler() {}
      public void handle(Callback[] callbacks)
            throws IOException, UnsupportedCallbackException
      {
            char[] password = …; // application specific
            for (Callback cb : callbacks)
            {
                  if (cb instanceof PasswordCallback)
                  {
                        ((PasswordCallback)cb).setPassword(password);
                  }
            }
      }
}
```

In iSM, a Keystore can also be used where a CertStore is expected. The corresponding Certstore contains the trusted certificate entries and the first certificate of each private key entries. For example, a Keystore can be used as TrustStore if it only contains the certificates of trusted CAs. A Keystore is not a general CertStore because it cannot contain a Certificate Revocation List.

The following table lists the Keystore Provider properties.

Property	Description
Name *	The name of the Keystore definition to add.
Description	A brief description of the use of this Keystore.
Keystore *	Location of the Keystore file or "NONE" if using PKCS11.
Keystore Password	The password used by the Keystore.
Keystore Type *	Keystore Type, for example, JKS or PKCS12, and so on.
Keystore JCE Provider	JCE Provider implementing this Keystore type.
Callback Handler	The fully qualified class name of a Callback handler that will satisfy authentication callbacks for the Keystore. The callback handler must satisfy the javax.security.auth.callback. CallbackHandler interface and be available on the iSM classpath.
Reload Period	Minimum time to wait before the provider checks if the Keystore needs to be reloaded. The format is [xxh][xxm]xx[s]. Enter 0 to check for reload every time the Keystore is requested. Leave the parameter empty to never reload the Keystore. A file based Keystore is reloaded only if the file was modified since last reload.

Directory CertStore Provider

A Directory CertStore provider implements a certificate store as a set of files in a file system directory. Each file in the directory is scanned for certificates or certificate revocation lists. A Directory CertStore provider is particularly useful in components that support CRL checking.

The provider accepts a sequence of DER-encoded certificates in binary or in printable base64. If the certificate is provided in Base64 encoding, it must be bounded at the beginning by —-BEGIN CERTIFICATE—-, and must be bounded at the end by —-END CERTIFICATE—-.

The provider also accepts a sequence of DER-encoded Certificate Revocation Lists in binary or in printable base64. If the CRL is provided in Base64 encoding, it must be bounded at the beginning by a line starting with —-BEGIN, and must be bounded at the end by a line starting with —-END.

The provider also accepts certificates and CRLs in pkcs#7 format.

The Reload Period property tells the provider how often to check whether the Certstore should be reloaded. By default, the Certstore is loaded only once the first time the provider is accessed and will never be reloaded. When defined, the Reload Period is the minimum time to wait before a reload can occur. The check occurs only when the provider is accessed so there is no cost if there is no activity. The value 0 means the provider must check for a possible reload every time it is accessed. The Certstore is completely reloaded if the directory or any of the files have modification times later than the last check. This guarantees additions and deletions are recognized.

The following table lists the Directory CertStore Provider properties.

Property	Description
Name *	The name of the Directory CertStore definition to add.
Description	A brief description of the use of this Directory CertStore.
CertStore Location *	CertStore directory location.
Certificate Factory JCE Provider	JCE Provider to use when creating the X.509 Certificate Factory.
Reload Period	Minimum time to wait before the provider checks if the directory contents was modified, hereby forcing the CertStore to be reloaded. The format is [xxh][xxm]xx[s]. Enter 0 to check the directory every time the CertStore is requested. Leave the parameter empty to never reload the CertStore.

SSL Context Provider

An SSL Context Provider defines the parameters used to make to make server or client connections secure over SSL. Once a provider is defined, it can be applied to IP-based protocols (such as HTTP or AS2). An SSL Context provider refers to other providers to simplify its configuration. You will need to create these providers before you can complete the creation of the SSL Context Provider.

The Security Protocol property specifies the version of the protocol. The options are: SSL, SSLv2, SSLv3, TLS, TLSv1, and TLSv2. SSL is the Secure Socket Layer and TLS is the Transport Layer Security. TLS is the successor of SSL. In fact, SSLv3 and TLSv1 are very similar except they are not compatible with each other.

TLS protocol version 2 is supported. Basic SSL is no longer considered to be sufficiently secure, and many of its shortcomings have been addressed with TLS. The SSL level setting represents the minimum acceptable security algorithm. iWay Software strongly recommends that TLS be considered as the minimum acceptable level. For secure transactions, specification of TLS version 2 is recommended, provided that both sides of the transaction have this algorithm available. iSM will negotiate for the highest level available when connections are established.

Note: You must have Java version 1.7 configured on your system to use TLS protocol version 2.

When the Client Authentication property is true, servers using this provider will use SSL client authentication, that is, the server must receive and authenticate a certificate from the client as part of the SSL handshake.

When the Hostname Verification property is true, clients using this provider will attempt to verify that the server's certificate matches its host name. That is, the common name part of the subject distinguished name must be the hostname.

The Enabled Cipher Suites property determines which cipher suites will be available during SSL negotiation. This is a comma-delimited list forming a subset of the Cipher Suites supported by the platform. If left blank, all available cipher suites will be enabled. A secure connection is only as secure as its weakest available cipher suite. Since some built-in cipher suites can be very weak, it is important to review the enabled cipher suites to verify they meet the security requirements of the application. The list of standard cipher suite names appears in Oracle's "Java Cryptography Architecture Standard Algorithm Name Documentation". The exact list of all cipher suites on the platform can be obtained by calling String[] SSLServerSocketFactory.getSupportedCipherSuites().

A Keystore provider is needed to know where to find the private keys. This can be the default SSL Keystore or a specific Keystore provider selected by name. A server always needs a private key associated with its server certificate. You can let JSSE pick a private key within the Keystore, or you can choose exactly which key to use by specifying the server key alias. A client does not need a private key associated with a client certificate, unless client authentication is used. In that case, you can let JSSE pick a private key within the Keystore, or you can choose exactly which key to use by specifying the client key alias.

Another Keystore provider is needed to list the certificate of Trusted CAs. Preferably, this should not contain any other keys because they would also be treated as Trusted CAs. In particular, it is not recommended to use the same Keystore provider for the private keys and the TrustStore. The client will use the TrustStore to validate the server certificate. The server will use the TrustStore to validate the client certificate when client authentication is used.

The validation can be improved by enabling certificate revocation to verify the peer certificate has not been revoked. This requires the name of a CertStore provider in the PKIX CertStore Provider property to specify where to find the certificate revocation lists.

An SSL Context maintains a session cache. Sessions in the cache can be reconnected with less overhead than those not cached. The Session Cache Size property determines the maximum number of SSL sessions that will be retained in the cache. The Session Timeout is the maximum length of time (in seconds) that an SSL session can remain in the cache.

It is possible to select exactly which JCE provider will be used to create certain objects. The JCE SSL Context Provider determines which provider will be used to create the SSLContext. The JCE PKIX Trust Manager Provider property specifies which JCE provider will be used to get the instance of the PKIX TrustManagerFactory. The JCE Signature Provider property tells the TrustManagerFactory which JCE provider to use to get Signature objects. These properties can be set to "Not Specified" to choose the corresponding default JCE provider.

The following table lists the SSL Context Provider properties.

Property	Description
Name *	The name of the SSL Context definition to add.
Description	A brief description of the use of this SSL Context.
Keystore Provider *	Configured Security Provider for the Keystore you wish to use for this SSL context. Choose *default* to use the default SSL Keystore Provider. Keystores hold private keys.
Truststore Provider *	Configured Security Provider for the truststore you wish to use for this SSL context. Choose *default* to use the default SSL Keystore Provider. Truststores hold the certificate of Trusted CAs used to verify peer certificates.

Property	Description
Security Protocol *	Specify the version of security protocol that should be used. During SSL handshake, a negotiation selects the protocol to be used from the best mutually supported. This field sets the minimum acceptable security protocol. If the handshake cannot select a mutually supported protocol, the connection fails. The options are: SSL, SSLv2, SSLv3, TLS, TLSv1, and TLSv2. **Note:** You must have Java version 1.7 configured on your system to use TLS protocol version 2.
JCE SSL Context Provider	JCE Provider for the SSL Context.
Server Key Alias	Alias for the key to be used to identify secure servers using this SSL context. If not supplied, the key will be selected using JSSE default behavior.
Client Key Alias	Alias for the key to be used to identify secure clients using this SSL context. If not supplied, the key will be selected using JSSE default behavior.
Session Cache Size	The maximum number of SSL sessions that will be retained in the session cache. Sessions in the cache can be reconnected with less overhead than those not cached.
Session Timeout	Maximum length of time (in seconds) that an SSL session can remain in the cache.
Enable Certificate Revocation	Enable CRL checking of certificates during handshake.
OCSP Responder	Name of the OCSP Responder provider. This verifies the status of certificates online instead of relying on Certificate Revocation Lists (CRLs).
JCE PKIX Trust Manager Provider	JCE provider to construct PKIX Trust Manager. Choose 'Not Specified' for default.
JCE Signature Provider	JCE provider used to verify digital certificate signatures during handshake.

Property	Description
PKIX Certificate Store	Certificate store from which certificate revocation lists are loaded.
Enabled Cipher Suites	If supplied, only cipher suites on this list will be enabled for SSL sockets or SSL engines created using this provider. The user must take care that enabled cipher suites are supported by other components specified. Enter as comma-delimited list or use FILE() function. If left blank, all available cipher suites will be enabled and be available during SSL negotiation.
Hostname Verification	If true, client SSL connections using this provider will attempt to verify that the server certificate matches its host name.
Client Authentication	If true, servers using this provider will use SSL client authentication, that is, the server must receive and authenticate a certificate from the client as part of the SSL handshake.

LDAP Certstore Provider

The LDAP CertStore Provider implements a CertStore by querying an LDAP server that exposes a schema adhering to RFC2587. Notice the Directory Provider used to access LDAP in a more general way can also be used to implement a CertStore. The LDAP CertStore provider is offered for those that need the extra configuration options made available by the Bouncy Castle LDAP CertStore. These options give a little flexibility to deviate slightly from RFC2587, though retaining the same working principles.

One difference between the LDAP CertStore provider and the Directory Provider is the Base DN has a separate property instead of being bundled in the URL.

When the Search For Serial Number In property is not null, the serial number of the certificate is searched in this LDAP attribute. The remaining properties allow you to give an alternate name for some of the LDAP attributes queried by the LDAP CertStore.

Property	Description
Name *	The name of the LDAP CertStore definition to add.

Property	Description
Description	A brief description of the use of this LDAP CertStore.
URL *	URL to reach LDAP directory. LDAP URL's are in the form ldap://*host[:port]*
Base DN	Base DN.
Search For Serial Number in	If not null the serial number of the certificate is searched in this LDAP attribute.
User Certificate Attribute	Attribute name(s) in the LDAP directory where end certificates are stored. Separated by space. Defaults to userCertificate.
CA Certificate Attribute	Attribute name(s) in the LDAP directory where CA certificates are stored. Separated by space. Defaults to CACertificate.
Cross-Certificate Attribute	Attribute name(s), where the cross certificates are stored. Separated by space. Defaults to crossCertificatePair.
Certificate Revocation List Attribute	Attribute name(s) in the LDAP directory where CRLs are stored. Separated by space. Defaults to certificateRevocationList.
LDAP User Certificate Attribute Name	The attribute name(s) in the LDAP directory where to search for the attribute value of the specified userCertificateSubjectAttributeName. For example, if cn is used to put information about the subject for end certificates, then specify cn. Defaults to cn.
LDAP CA Certificate Attribute Name	The attribute name(s) in the LDAP directory where to search for the attribute value of the specified CACertificateSubjectAttributeName. For example, if ou is used to put information about the subject for CA certificates, then specify ou. Defaults to: cn ou o.
LDAP Cross-Certificate Attribute Name	The attribute name(s) in the LDAP directory where to search for the attribute value of the specified crossCertificateSubjectAttributeName. For example, if o is used to put information about the subject for cross certificates, then specify o. Defaults to: cn ou o.

Property	Description
LDAP Certificate Revocation List Attribute Name	The attribute name(s) in the LDAP directory where to search for the attribute value of the specified certificateRevocationListIssuerAttributeName. For example, if ou is used to put information about the issuer of CRLs, specify ou. Defaults to: cn ou o.
User Certificate Subject Attribute Name	Attribute(s) in the subject of the certificate which is used to be searched in the ldapUserCertificateAttributeName. For example, the cn attribute of the DN could be used. Defaults to cn.
CA Certificate Subject Attribute Name	Attribute(s) in the subject of the certificate which is used to be searched in the ldapCACertificateAttributeName. For example, the ou attribute of the DN could be used. Defaults to: o ou.
Cross-Certificate Subject Attribute Name	Attribute(s) in the subject of the cross certificate which is used to be searched in the ldapCrossCertificateAttributeName. For example, the o attribute of the DN may be appropriate. Defaults to: o ou.
Certificate Revocation List Issuer Attribute Name	Attribute(s) in the issuer of the CRL which is used to be searched in the ldapCertificateRevocationListAttributeName. For example, the o or ou attribute may be used. Defaults to: o ou.

OCSP Responder Provider

The Online Certificate Status Protocol (OCSP) is an Internet protocol used to obtain the revocation status of an X.509 digital certificate. It is formalized in RFC 2560.

OCSP was created as an alternative to Certificate Revocation Lists (CRLs), specifically addressing certain problems associated with using CRLs in a Public Key Infrastructure (PKI).The request and response nature of these messages lead to OCSP servers being termed OCSP responders. iSM can communicate with an OCSP responder to obtain the revocation status of a certificate, avoiding the need to manage certificates locally in many cases.

The following table lists the OCSP Responder Provider properties.

Property	Description
Name *	The name of the OCSP Responder definition to add.
Description	A brief description of the use of this OCSP Responder.
Responder URL *	Location of the OCSP responder. For example: `http://ocsp.example.net:80`
Certificate Subject Name	Subject name of the certificate for the OCSP responder. For example, CN=OCSP Responder and O=XYZ Corp.
Certificate Issuer Name	Issuer name of the certificate for the OCSP responder. For example, CN=Enterprise CA and O=XYZ Corp. This property is required if a value for the Certificate Subject Name parameter is not specified.
Certificate Serial Number	Serial number of the OCSP responder's certificate. For example, 1234567890123456789. This property is required if a value for the Certificate Subject Name parameter is not specified.
Certificate Store *	Certificate store where the responder certificate can be retrieved.
HTTP Client Provider *	HTTP client provider that manages outgoing connections to the responder.

Chapter 4

Configuring Runtime Security Using Access Control

This section describes how to configure run-time security using access control.

In this chapter:

Runtime Security Overview

iWay Service Manager (iSM) offers runtime security capabilities for highly secure applications. Runtime security includes *authentication*, often a logon process that validates a user's credentials, and *authorization*, in which execution depends on the permissions granted to the user.

Protocol based logon may involve a scheme, that is, a standardized method by which a server requests credentials and a client offers them. In iSM, the validation of credentials is handled by *authentication realms*. If a user is authenticated, the realm returns a *principal*, a structure that holds the user's credentials along with a list of the *security roles* assigned to the user. Once established, the principal is available for the duration of the transaction or session.

In addition to the creation of principals by logon, iSM provides services to define and assign a principal to the current transaction. If the transaction already has one or more associated principals, the new principal masks these. This is sometimes called impersonation. Additionally, a process that uses its own logic to identify a user can create a principal for use in the remainder of the transaction or session.

An application determines whether a particular action is authorized by checking to see whether the current principal has a role that has been granted the permission required for that action. The mapping of roles to permissions is sometimes called an *Access Control List*, or ACL.

The main components of iSM runtime security are:

1. Logon Schemes
2. Realm Based Authentication
3. Role Based Authentication
4. Impersonation

Logon Schemes

A logon authentication scheme is a protocol that defines the challenge sent by the server in response to a request for a secure resource and the exchange of security information between client and server in response to the challenge. Currently, the NHTTP and NAS2 listeners support two such authentication schemes, HTTP Basic and Digest Access authentication, as defined in RFC 2617. In addition to these, the NHTTP emitter and HTTP client provider support NTLM and the negotiate scheme using Kerberos and SPNEGO. For more information on how to configure the emitter to work with negotiate and Kerberos, see *Authenticating an HTTP Client Using Kerberos* on page 209.

Authentication on the NHTTP listener can be tested using browsers that can show the request and response HTTP headers.

With basic authentication, you should see:

1. The server responds to the initial request with 401 status code and a challenge. For example:

   ```
   WWW-Authenticate: Basic realm="realm name"
   ```

   The realm name in our challenges will always match the name of the configured realm in the server.

2. The client responds to the challenge by adding a header to its request. For example:

   ```
   Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
   ```

   The secret looking string after "Basic" contains the user name and password with Base-64 encoding.

   If you are using digest authentication, the challenge and response have the following structure:

   - **Challenge**

     ```
     WWW-Authenticate: Digest realm="realm name",
                       qop="auth",
                       nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                       opaque="5ccc069c403ebaf9f0171e9517f40e41"
     ```

The server, desiring credentials, passes the request to the client along with a newly-created random number, called a nonce. The client hashes this nonce along with the user ID, the realm name and the password (plus a few other things) to create a message digest containing the hash code.

- **Response**

  The server performs the same calculation, but using its stored password.

```
Authorization: Digest username="Mufasa",
                      realm="realm name",
                      nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                      uri="/dir/index.html",
                      qop=auth,
                      nc=00000001,
                      cnonce="0a4f113b",
                      response="6629fae49393a05397450978507c4ef1",
                      opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

  The response should match the value computed by the server. The actual password never appears on the line.

Other protocols obtain their logon user ID and password based on the RFC controlling their operation. The FTP Server channel and the Telnet command channel are cases in point.

Configuring Credential Requirements

A user is authorized to perform an operation only if that user is assigned with a role that has been granted the required permission for that operation. Users and their roles are configured in the authentication realm. When a realm authenticates a user, it creates a Principal that contains all of the security roles for the user. The list of roles that have been granted a particular permission is called an Access Control List (ACL). Assigning roles to users and granting permissions to roles is the responsibility of the security officer.

To execute an iWay Service Manager (iSM) operation, a user must either have the *ism.admin* role, <*configname*>*.admin* role, or another role assigned that has been granted the permission required for that command. The names of the specific permissions for iSM commands are built into the server.

The `set acl` command grants a permission to a role and assigns the role to either the general configuration or an application/configuration. If the role applies to a specific application/configuration, then the permissions associated with that role in the application/configuration are used. If the role is not applicable to the specific application/configuration, then the permissions associated with the role in the general server are used. For example, the *devmgr* role might offer the ability to control a development application, but deny any permissions for production applications other than review.

When using the `set acl` command to grant a permission to a role in an application, specify the configuration by name as follows:

```
set acl devserver/devmgr  cmdstop  -append
```

Roles can be assigned to server groups by use of regular expressions. For example, if your installation named all development servers with the prefix `DEV_`, then you might use the `set acl` command as follows:

```
set acl DEV_./devmgr cmdstop  -append
```

The regular expression `DEV_.` indicates all servers beginning with the `DEV_.` prefix.

When checking permissions, the security system of iSM checks in the following order:

1. Specific configuration roles.
2. Regular expression configuration roles.
3. General server roles.

Note: The general permission *ism.admin* must be held by the user in order to use the `set acl` or `set policy` commands via the command line or the iSM Administration Console.

For example, to issue the start command, a user must have the required permission. The ACL for the start command is named *cmdstart*. For more information, see *iSM Commands and Corresponding ACL Names* on page 39. In this scenario, a security officer has decided that a user with the permission *starter* can issue the start command. The ACL for the stop command is named *cmdstop*. In addition, the security officer has decided that the permission *starter* can also stop. To accomplish this, once when the server is installed, the security officer (with administrative authority) must issue the following commands:

```
set acl cmdstart starter
```

```
set acl cmdstop starter
```

At some point the security officer may decide to grant a user with the permission, *startonly*, the ability to start a channel, but not to stop a channel. The security officer issues the following command:

```
set acl cmdstart startonly
```

Next, the security officer creates an authentication realm. For more information on the authentication realm, see *Realm Based Authentication* on page 40.

For this example, a properties realm is defined (users.properties), which is commonly used for simple situations. The security officer creates two users, each with a name and password. Tom (password=tomspassword) can start and stop channels, but Fred (password=fredspassword) can only start channels.

The security officer adds the following settings to the properties realm (users.properties):

```
tom=tomspassword
tom.role0=starter
fred=fredspassword
fred.role0=startonly
```

Additionally, the security officer decides that Tom can also run process flows from the command line. The security officer issues the following command:

```
set acl cmdflow flower
```

The security officer also adds the following line to the properties file:

```
tom.role1=flower
```

iSM Commands and Corresponding ACL Names

The roles of the command handler are listed in the following table.

iSM Command	ACL Name
Enqueue	cmdflow
Flow	cmdflow
Pull	cmdpull
Refresh	cmdrefresh
Remote	cmdremote
Run	cmdrun
Set acl	cmdsetacl
Set policy	cmdsetpolicy
set property	cmdsetproperty
Set register	cmdsetregister
Shell (or !)	cmdsys
Start	cmdstart

iSM Command	ACL Name
Stop	cmdstop

Realm Based Authentication

Once the application has obtained the credentials from the user, they must be verified and the user's security roles must be determined. In iSM, this function is performed by authentication realms. A realm represents a database of information about valid users of the system, containing, at a minimum, the user ID, password, and the names of any security roles that have been assigned to the user. iSM supplies six different realm implementations, which can be configured in the Provider section of the iSM Administration Console.

1. **Properties Realm**

   This realm is configured with the path to a properties file that contains properties such as *username=password* and *username.role=rolename*. For example:

   ```
   steve=password
   steve.role0=user
   judy=secret
   judy.role0=admin
   judy.role1=bigdocs
   clement=Canada
   clement.role0=admin
   richard=masked
   richard.role0=financial.admin
   richard.role1=audit.admin
   ```

   In this example, *judy* has two access tokens, while *steve* has one. Unlike *steve*, *judy* can work with *bigdocs*. The meaning of *bigdocs* is subject to tests in the process flow, and is not defined by iWay.

   Clement is a general administrator, while *richard* has been granted administrative privileges for two applications (*financial* and *audit*). Obviously, this provides minimal security and is intended for use only during testing and debugging. However, this realm does show how roles are related to users.

2. **Console Realm**

   This realm wraps the iSM Administration Console security systems, authenticating users that have been defined on the Managed Servers page of the console. If the user has been granted power user rights, the Principal returned will include the admin role. The console realm is not configured as a provider. An instance of the console realm is always available to the system under the name, *consolerealm*.

3. **JDBC Realm**

   This realm authenticates users using a JNDI data source, which could be an iWay JDBC Provider. The following table lists and describes the configuration parameters for the JDBC authentication realm:

Parameter	Description
JNDI Factory Name	Initial Context Factory class to access the data source using JNDI. To use an iWay JDBC provider, enter *com.ibi.jndi.XDInitialContextFactory*.
JNDI Name	Name for the data source in the JNDI directory. For iWay JDBC providers, use the following format: `jdbc/[provider_name]`
Users Table	Table with at least one row for each valid user in this realm. The table must include at least two columns, containing user name and password. The names of these columns are specified by the Username and User Credential Column parameters. The table may include other columns if the application requires them.
User Roles Table	Table with at least one row for each security role assigned to a user. The table must contain at least two columns, for user name and role. The names of these columns are specified by the Username and Role Column parameters. The table may include other columns if required by the application.
Username Column	Name of the column that contains the user name in the Users and User Roles tables.
User Credential Column	Name of the column that contains the password in the Users table.
Role Column	Name of the column that contains the name of the user security role in the User Roles table.

The simplest tables that could be used with this realm might look as follows:

```
CREATE TABLE users
(
username varchar(25),
password varchar(25)
)
CREATE TABLE user_roles
(
username varchar(25),
role varchar(25)
)
```

To look up the password for a user, the realm constructs an SQL statement. For example:

```
SELECT password FROM users WHERE username = ?
```

And it finds roles with the following statement:

```
SELECT role FROM user_roles WHERE username = ?
```

4. **JAAS Realm**

   This realm wraps a JAAS logon context that must be configured externally in the JAAS logon configuration file. The JAAS realm does not support digest authentication. The following table lists and describes the configuration parameters for the JAAS authentication realm:

Parameter	Description
JAAS Config File	Path to the standard JAAS configuration file. If the system variable *java.security.auth.login.config* is not set when the realm is initialized, the realm will set it to this value. Consult JAAS documentation for the structure of this file.
Application Entry	The entry in the JAAS configuration to which this realm will delegate authentication requests.
User Principal	A successful JAAS logon results in a Subject that can contain one or more Principal objects. Specify the class name of the Principal object that should be understood as representing the authenticated user.

Parameter	Description
Role Principal	Specify the class name (or a comma delimited list of class names) for Principal objects created by the JAAS logon that should be understood as representing security roles for the authenticated user. The value returned by the Principal's getName() method will be the name of the role assigned to the user in iSM.

For testing, a simple JAAS logon module is provided that wraps the iSM console user database. To try it, you must first create the JAAS configuration file with an application entry pointing to the console module. For example:

```
test
{
com.ibi.providers.auth.jaas.ConsoleLoginModule REQUIRED;
};
```

Configure the JAAS authentication realm according to the values that are listed in the following table:

Parameter	Value
JAAS Config File	The file containing the application entry above.
Application Entry	For the above example, "test".
User Principal	The logon module creates a principal for the user of type: `com.ibi.providers.auth.jaas.ConsoleUserPrincipal`
Role Principal	The logon module creates a principal for the role of type: `com.ibi.providers.auth.jaas.ConsoleRolePrincipal`

It should be possible to configure the JAAS Realm to work with most logon modules.

5. **LDAP Realm**

This realm implementation works with a directory server accessed via the Java Naming and Directory Interface (JNDI) APIs. The following table lists and describes the configuration parameters for the LDAP authentication realm:

Parameter	Description
LDAP Provider	Name of the directory provider describing the connection to the LDAP server.
User Base Context	The base of the subtree containing users. Each user that can be authenticated must be represented by an individual entry that corresponds to an element in this directory context. If no value is specified, then the top level element in the directory context will be used.
User Pattern	A pattern for the distinguished name (DN) of the user's directory entry. Use {0} to substitute the user name. For example, (cn={0}). LDAP OR syntax is also supported. For example, (|(cn={0})(cn={0},o=myorg)). You can use this parameter instead of User Search Filter, Search User Subtree, and User Base Context when the distinguished name contains the user name and is otherwise the same for all users.
Search User Subtree	The search scope. Set to *true* if you wish to search the entire subtree rooted at the User Base Context entry. The default value of *false* requests a single-level search including only the top level.
User Search Filter	The LDAP filter expression to use when searching for a user's directory entry, with {0} marking where the actual user name should be inserted. Use this parameter (along with the Search User Subtree parameter) instead of the User Pattern parameter to search the directory for the user's entry.

Parameter	Description
User Password Attribute	Name of the attribute in the user's entry containing the user's password. If you specify this value, then this realm will retrieve the corresponding attribute for comparison to the value specified by the user being authenticated. If you do not specify this value, then this realm will attempt a simple bind to the directory using the distinguished name (DN) of the user's entry and password specified by the user, with a successful bind being interpreted as an authenticated user.
Role Base Context	The base directory entry for performing role searches. If no value is specified, then the top level element in the directory context is used.
Search Role Subtree	Set this parameter to *true* if you want to search the entire subtree of the element specified by the Role Base Context for role entries associated with the user. The default value of *false* causes only the top level to be searched.
Role Search Filter	The LDAP filter expression used for performing role searches. Use {0} to substitute the distinguished name (DN) of the user, and/or {1} to substitute the user name. If no value is specified, then a role search does not take place and roles are taken only from the attribute in the user's entry specified by the User Role Attribute parameter.
Role Attribute	The name of the attribute that contains role names in the directory entries found by a role search. In addition you can use the User Role Attribute parameter to specify the name of an attribute, in the user's entry, containing additional role names. If no value is specified, then a role search does not take place, and roles are taken only from the user's entry.

Parameter	Description
User Role Attribute	The name of an attribute in the user's directory entry containing zero or more values for the names of roles assigned to this user. In addition you can use the Role Attribute parameter to specify the name of an attribute to be retrieved from individual role entries found by searching the directory. If no value is specified, then all the roles for a user derive from the role search.

6. **Active Directory Realm**

   The Active Directory realm authenticates the user with a login to Active Directory. The properties are identical to the LDAP realm, except that the User Password Attribute parameter is not used to force a login. This realm displays a warning on the General Properties page of the iSM Administration Console if the password is about to expire in Active Directory, as shown in the following image.

LDAP Configuration for iWay Service Manager Administration Console Authentication

The new feature for LDAP authentication provides additional capability for iWay Service Manager (iSM) to authenticate against LDAP and associate an LDAP iSM role to the user.

iSM includes a built-in role for an administrator that allows for complete management and control of iSM from the Administration Console. Other roles may be added from the Web Console to limit access and management of iSM.

To implement LDAP authentication for the iSM Administration Console, each of these roles need to be added to an LDAP/Active Directory configuration as Groups and then associated to users. Optionally, an LDAP attribute like title may be associated to a role like ism.admin.

The built-in administrator roles are:

- ism.admin
- <*configname*>.admin

  For example, myapp.admin for an application/configuration that is named myapp.

Additional roles will need to be defined in the iSM Administration Console and also in LDAP. The following procedure describes the required configuration for LDAP and iSM.

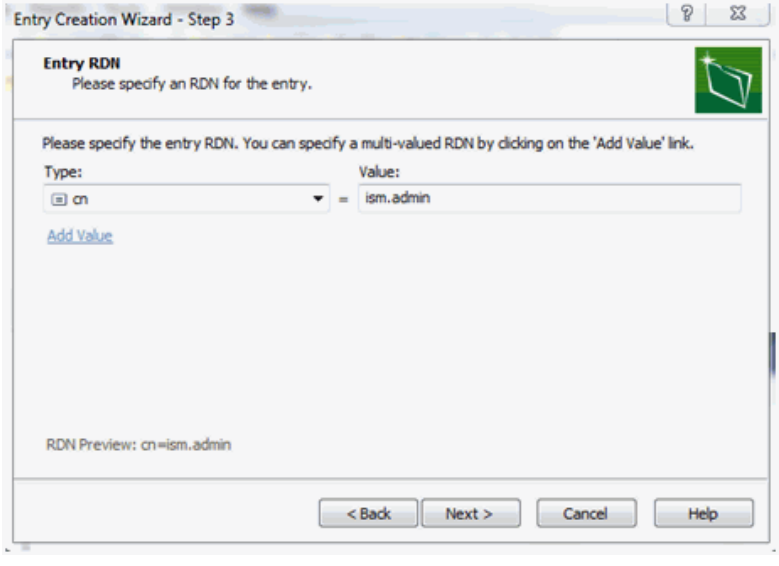
LDAP Setup and Configuration

This section describes how to set up an LDAP group, how to setup and configure iWay Service Manager (iSM), and how to add additional Server roles in iSM.

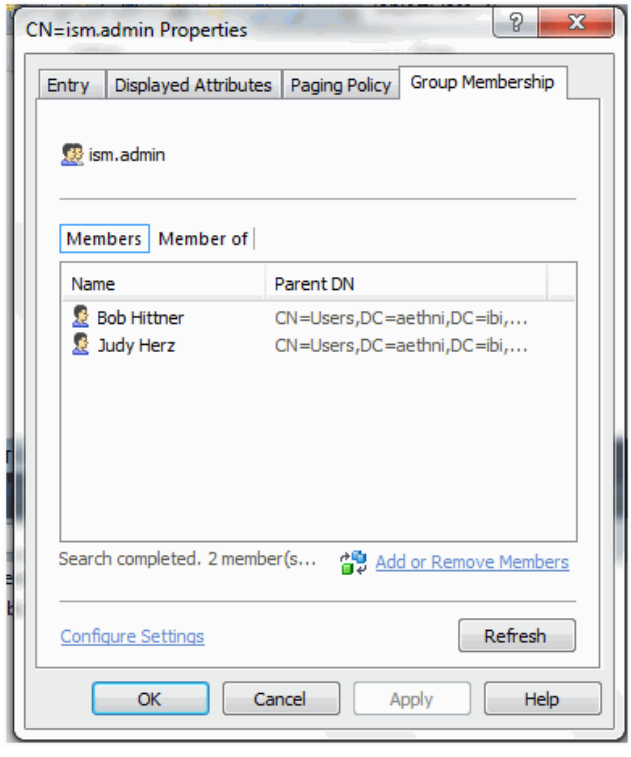
Procedure: How to Setup the LDAP Group

This section describes the LDAP group setup for iSM roles. The following steps describe creating an LDAP group based on the iWay Service Manager role and a common name (cn) that is the iSM role. Subsequently adding LDAP members to the group. For example, the LDAP group for the built-in iSM administrator, ism.admin will have a CN=ism.admin.

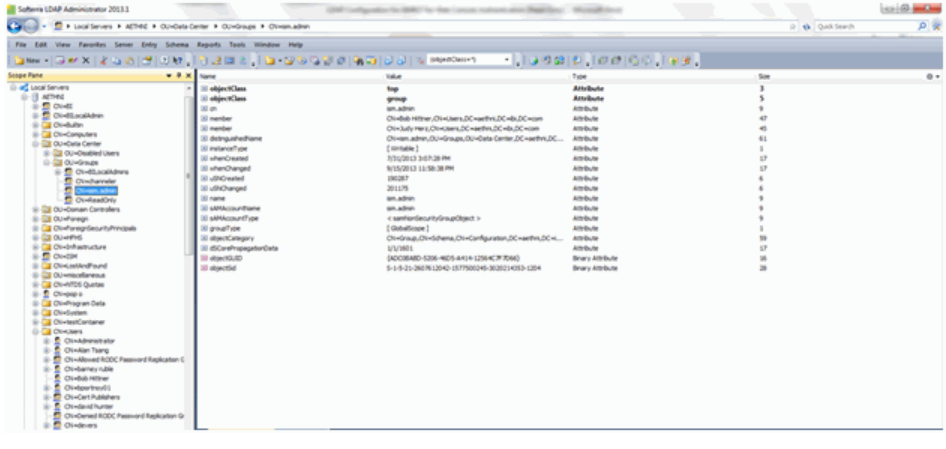
1. Create a new LDAP Group for ism.admin and set `cn=ism.admin`, as shown in the following image.



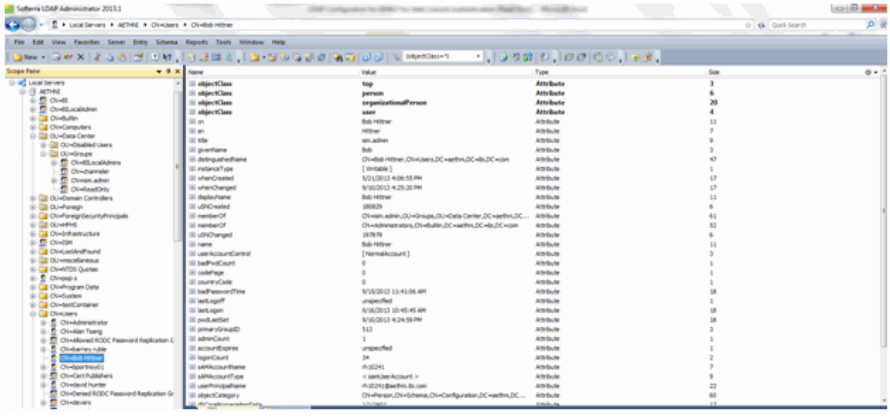
2. Add LDAP users as members to the LDAP Group *ism.admin*, as shown in the following image.



The LDAP Group *ism.admin* and the associated members of the group can be viewed in the scope pane, as shown in the image below.



LDAP User screen showing the example used earlier, Bob Hittner, is a member of the *ism.admin* LDAP Group, is displayed in then image below.

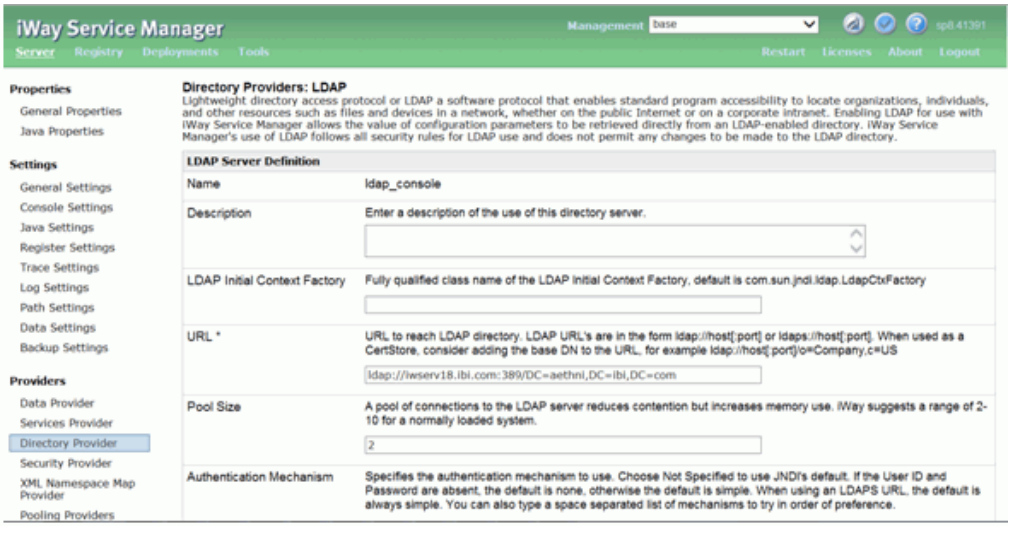


Procedure: How to Setup and Configure iWay Service Manager

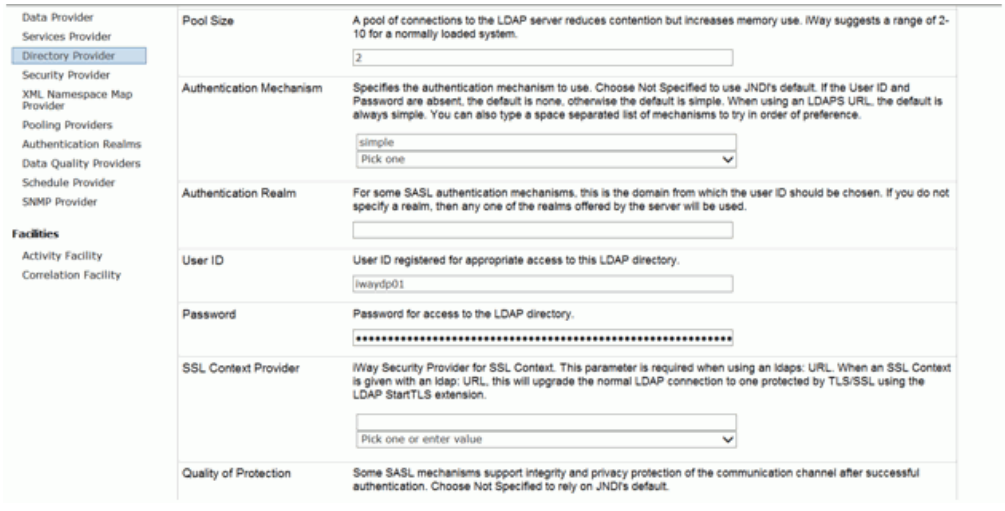
The following section describes the steps to configure iWay Service Manager (iSM) to access and authenticate against LDAP. The following steps describe how to create an iSM Directory provider and an authentication realm.

1. To create an iSM Directory Provider to access the LDAP directory, login to the iSM Administration Console. Click the *Server* Link, then click *Directory Provider.*

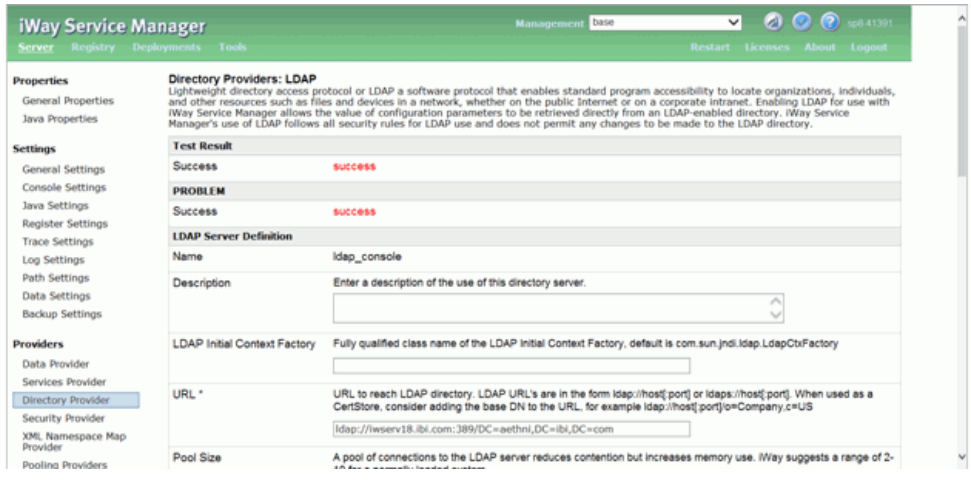
   The Directory Providers: LDAP dialog is displayed, as shown in the following image.



2. Update all the required fields for the LDAP Directory Provider, including the LDAP *URL*, and *Base DSN* fields.
3. Click the Authentication Mechanism drop-down and select *Simple*. Add the User ID and Password for LDAP, as shown in the following image.

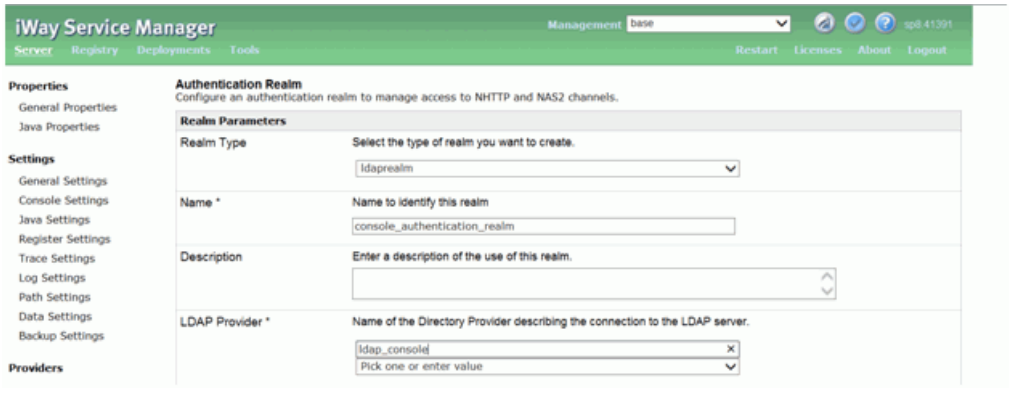


4. Click *Add*. Test for successful connectivity. If the connection is successful, the following dialog is displayed.

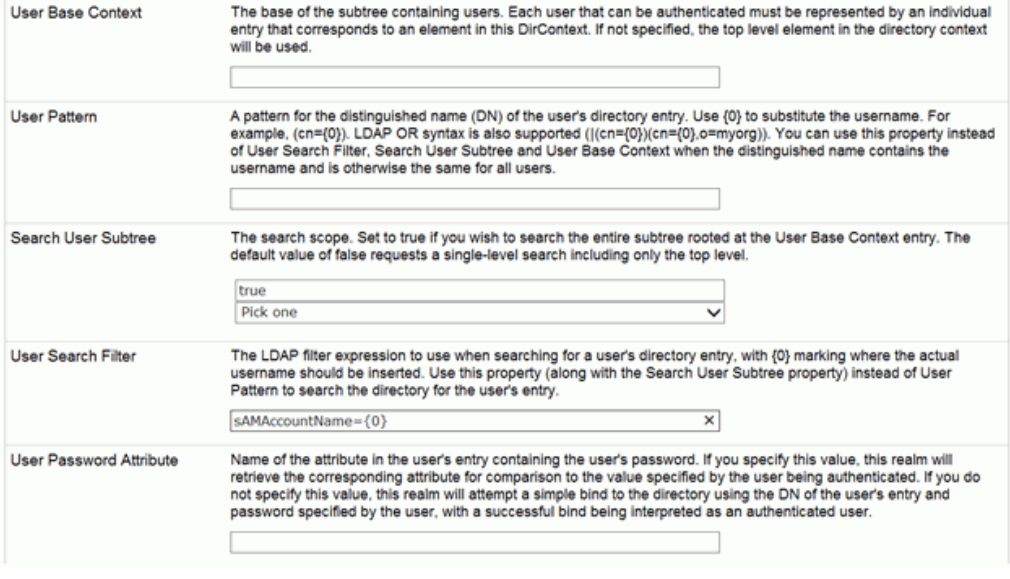


5. To add the Authentication Realm, click *Authentication Realm* under Providers.
6. Click *New*.

7. In the Authentication Realm dialog, Select *ldaprealm* from drop-down in the Realm Type field. Enter a name, and select the configured directory provider in the LDAP Provider field, as displayed in the following image.



8. in the Search User Subtree drop-down, select *true*.
9. In the User Search Filter field, enter s*AMAccountName={0}*, as shown in the following image.



10. Enter Role based information as follows:

    - ❏ Role Base Context: *OU=Groups,OU=Data Center*.
    - ❏ Search role Subtree: Select *true* from the drop-down.

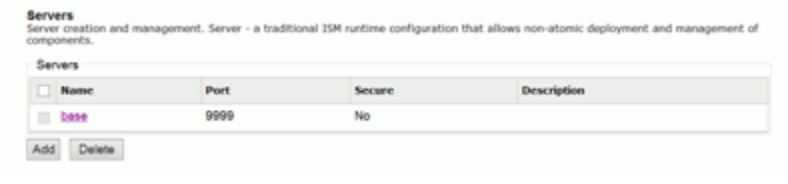
- Role Search Filter: Enter *member={0}*.
- Role Attribute: Enter *cn*.

11. Click *Finish*.
12. To update iSM Console Security, Click *Management*, as shown in the following image.

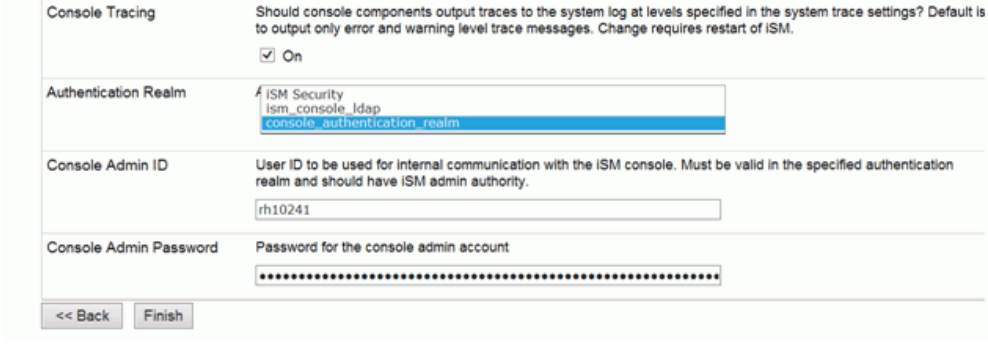
13. Click *Servers*.

    The Server creation and management dialog is displayed.

14. Click the configuration name that will be using the LDAP authentication, as shown in the following example.



15. Under Console Attributes, update the Authentication Realm parameter to use the authentication realm created above, as shown in the following image.



16. Enter the LDAP User ID and Password that is associated with *ism.admin* LDAP Group.

    Optionally, check Console Tracing to debug LDAP authentication issues.

17. Click *Finish*.
18. Restart iSM and login to the iSM Administration Console.

Note: To debug LDAP authentication problems, start iSM from a Windows Command Prompt window and start iSM using the following command and option -u; iway61 base -u.

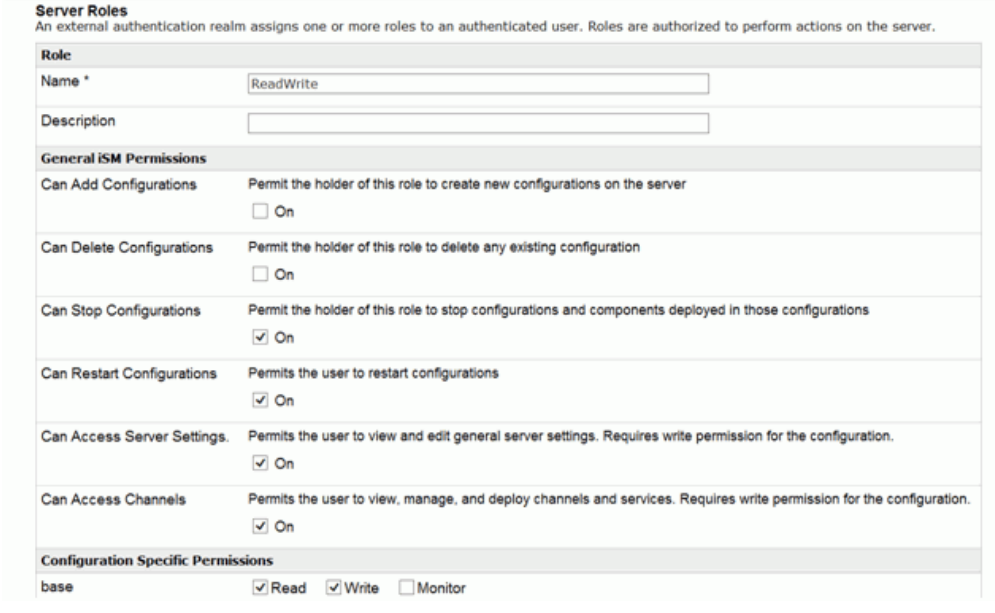
The setup of LDAP connection and authentication for iSM is complete.

Procedure: How to Add Additional Server Roles in iWay Service Manager

The following section describes how to define additional iSM roles.

1. Click *Management* and then *Server Roles*.
2. Click *Add*.

   The Server roles page is displayed.
3. In the Name field, enter *ReadWrite*.
4. Select the allowable actions, as shown in the following image.



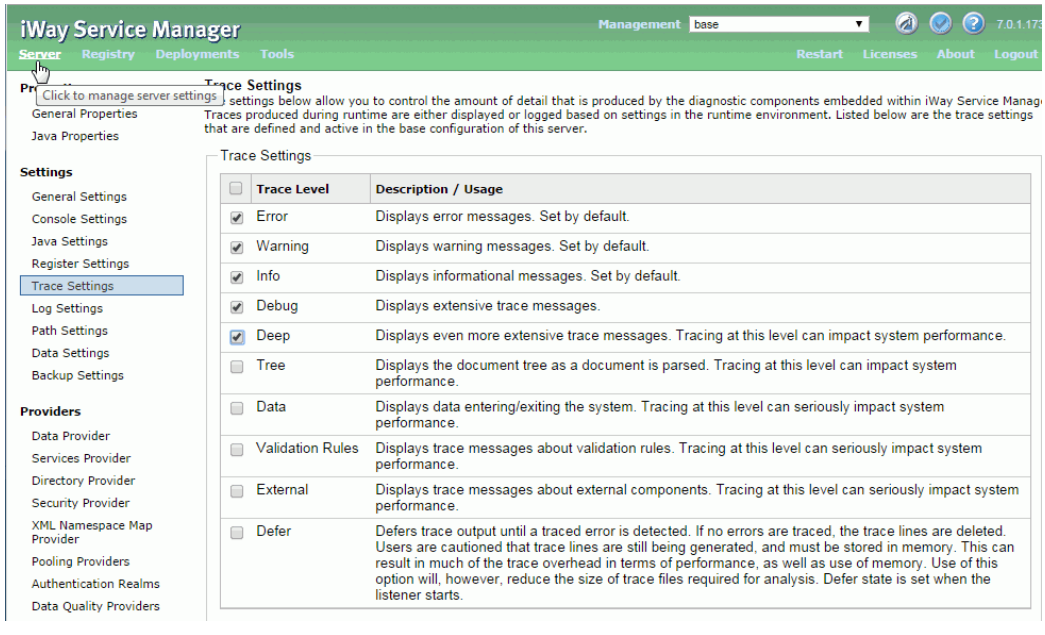
5. Follow the steps under *LDAP Setup and Configuration* on page 47 to add the LDAP Group for the *ReadWrite* role.

   Note: The name of the iSM role must match the name of the LDAP group i.e. iSM role = *ReadWrite* LDAP Group = *ReadWrite*. They are mapped by using the same name.
6. Restart iSM and verify new role.

Troubleshooting LDAP Authentication in iSM configuration log(tracing)

To troubleshoot LDAP authentication in iSM, perform the following:

- Enable Console tracing as described in step 16 of *How to Setup and Configure iWay Service Manager* on page 50.
- Click on the Server link and click Trace Settings. Enable *Debug* and *Deep*, as shown in the following image.



- Try logging in to the iSM console(configuration)
- Collect log files from the iSM config log directory in the following path: `\\iwayhome\config\base\log`. Review the most recent log file.

  For example:

```
DEEP (console)   LDAP Realm, entry found for csswxz with dn
CN=CSSWXZ,CN=Users,DC=eda,DC=csseda,DC=com
DEEP (console) LDAP User role name cn search
DEEP (console)  LDAP Realm, retrieving values for attribute cn
DEBUG (console) LDAP Realm, csswxz authenticated successfully
DEEP (console)  LDAP Realm,
getRoles(CN=CSSWXZ,CN=Users,DC=eda,DC=csseda,DC=com)
DEEP (console)  LDAP Realm, retrieving values for attribute cn
DEEP (console)  LDAP Realm, Returning roles: CSSWXZism.admin
DEEP (console) LDAP Realm, Closing directory contex
```

Note: The ism.admin role may seem joined to another role(CSSWXZism.admin). This is expected behavior. Check to make sure that the ism.admin role, or any other role you assign, exists under *Returning roles* in the log file.

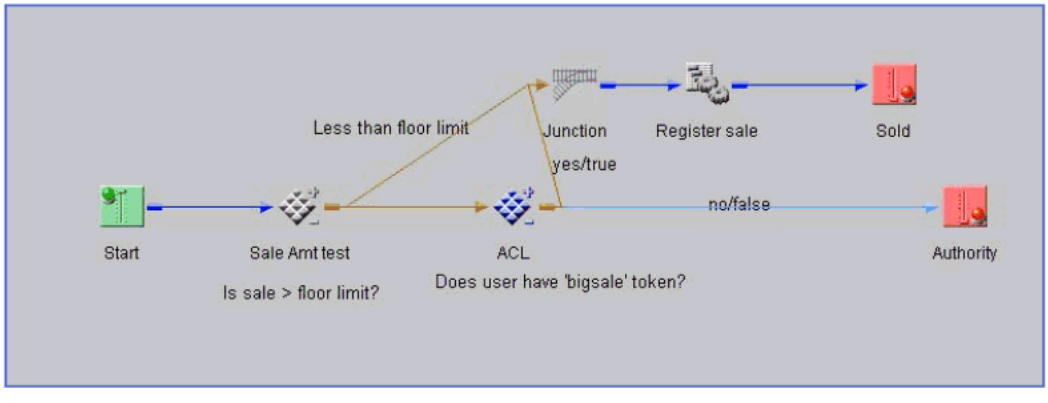
Role Based Authentication

Roles are represented by tokens carried in the principal. Roles are used within the process flow to control the level of authority needed to perform specific actions. Roles can be tested within the flow by the iWay Functional Language expression _hasrole(name).

The _hasrole() function returns true if the current principal carries the named token, otherwise it returns false. For more information about this function and other security-related iFL functions, see the *iWay Service Manager Component and Functional Language Reference Guide*.

A sample use might be to test the role to determine whether a leg of the process flow is permitted. A standard test object can be employed for this purpose. The true edge is followed if the role is present, otherwise the false edge is followed.

In this example, a document arrives to register a sale. If the amount of the sale is under a selected floor limit, the sale is registered. If greater than the floor limit, the sender must have the 'bigsale' token. This is tested in the test object named ACL.



Impersonation

Impersonation refers to the creation of a new principal, carrying user ID and password along with any desired roles. The principal is created using the Authenticate/Impersonate service (XDPrincipalAgent). Once created, it is used for all subsequent actions.

You cannot create a principal with the admin role unless the principal currently in force has that role.

Principals can be used to control the authority passed onward in a flow, possibly to a subflow. The principal will be carried to a channel reached via the internal emit service, so that the channel inherits the authorizations of the calling channel.

The Authenticate/Impersonate service (XDPrincipalAgent) can either create a principal or destroy the last one that it created. For more information on how to configure this service, see *Authenticate/Impersonate Service* on page 196.

Chapter 5

Realm-based Security in the iSM Administration Console

This section describes how to configure realm-based security in the iSM Administration Console.

In this chapter:

Realm-based Security Overview

Access to the iSM Administration Console can be configured to use any authentication realm as an alternative to the built-in iSM security (or console realm). This allows the management of iSM console users to be integrated with existing security systems.

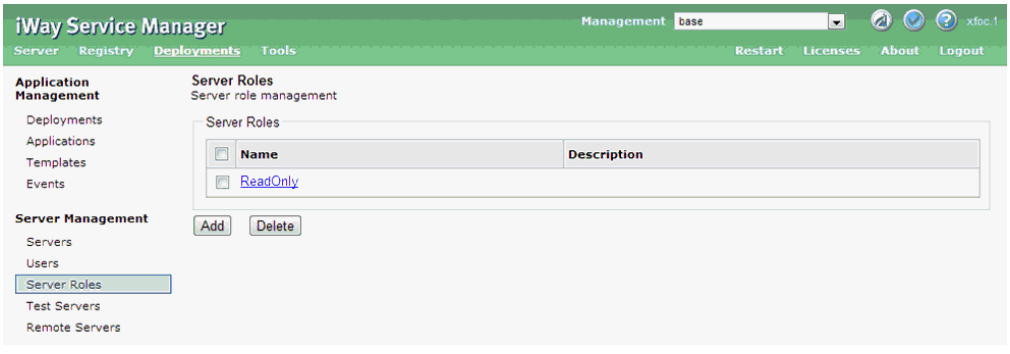
The authentication realm performs two tasks:

- **Authenticate credentials of the user.** This determines whether or not the user is allowed access to iSM. The iSM Administration Console uses the HTTP Basic Authentication scheme.
- **Return the list of security roles assigned to the user.**

A security role can be understood as a particular function that a user performs in iSM. The specific permissions that are necessary for this function are granted to the role, and the role is assigned to a user. To determine whether the user is authorized to do something in the iSM console, the system checks to see whether the user has a role with the required permissions. A user can be assigned multiple roles, since the permissions assigned to these roles are cumulative.

Creating or Editing a Role in the Management Section of the Console

The console access permissions that can be granted to a role are identical to those that can be granted to a user in the iSM console security scheme. There are general permissions, applicable to all iSM configurations, supplemented by read, write, and monitor permissions for each individual configuration. A realm user with a role that has been granted the same permissions as an iSM Security user should have the same level of access to console services as the iSM user.



Changes to roles do not take effect until iSM is restarted. You cannot create the Administrator role here. The Administrator role is built into the system, with the name "ism.admin". A user with the ism.admin role has full access to all iSM console services.

The iWay Service Manager offers several different types of authentication realm that can be used for console security. Security roles must be assigned to users within the realm. For more information on how to configure each type of realm and assign roles, see *Realm Based Authentication* on page 40 and *Role Based Authentication* on page 56.

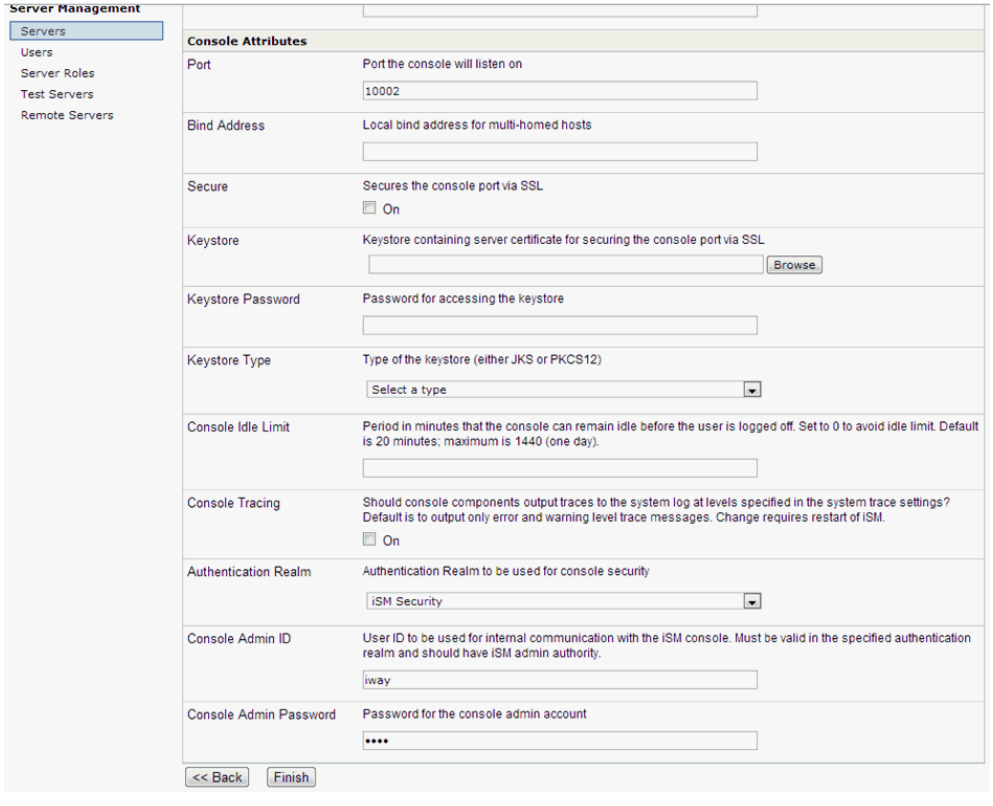
To enable realm-based authentication for an iSM configuration, you must specify the realm to use in the properties of that configuration, on the Servers page in the Management section of the console.

The following table lists and describes the three conditions that must be set.

Parameter	Description
Authentication Realm	Specifies the realm that will be used for console authentication for this server. Note that the realm must be configured in the server where you intend to use it. Select *iSM Security* for the original, user-based, security scheme.

Parameter	Description
Console Admin ID	iSM often exchanges background messages between configurations using the console service. This parameter specifies the user ID that should be sent with these internal requests. This user must be valid in the authentication realm you specify for this configuration.
Console Admin Password	The password to use with the Console Admin ID for background requests to the console service of this configuration.

The following image shows the Console Attributes pane in the iSM Administration Console.



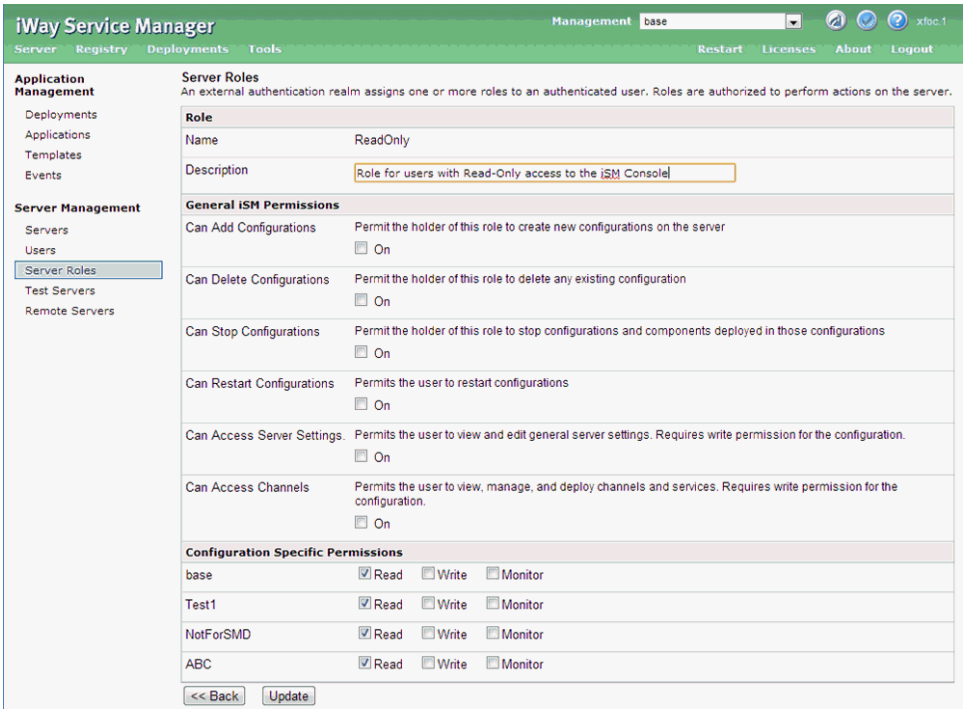
Changes to the authentication realm of the configuration or console administration ID will not take effect until the console is restarted.

You can force iSM to revert to *iSM Security* by starting the server from the command line using the -u switch. This is useful to correct any problems that occur while configuring realm-based authentication.

Configuring Authentication for a Base Configuration Using a Properties File Realm

This section provides an example that demonstrates how authentication for the base configuration can be configured using a properties file realm. The properties file realm is the simplest authentication realm that iSM offers, and is intended for prototypes and demonstration, not production use. A role is created for users who have read-only access to the console, and define two users in the properties realm, an administrator, and a user with the read only role. The properties realm in iSM and the base configuration to use the realm for console authentication will then be configured.

1. In the left console pane of the Deployments menu, select *Server Roles*, and then click *Add* to add a new role.
2. Provide a name for the new role (for example, ReadOnly), and then click all the *Read* check boxes in the Configuration Specific Permissions section. Do not provide any other permissions, as shown in the following image.



The properties realm uses a simple Java properties file to define users and their roles.

3. Create the following text file:

```
Admin=password
Admin.role0=ism.admin
User=password
User.role0=ReadOnly
```

4. Save the properties file as *consoleauth.properties*.

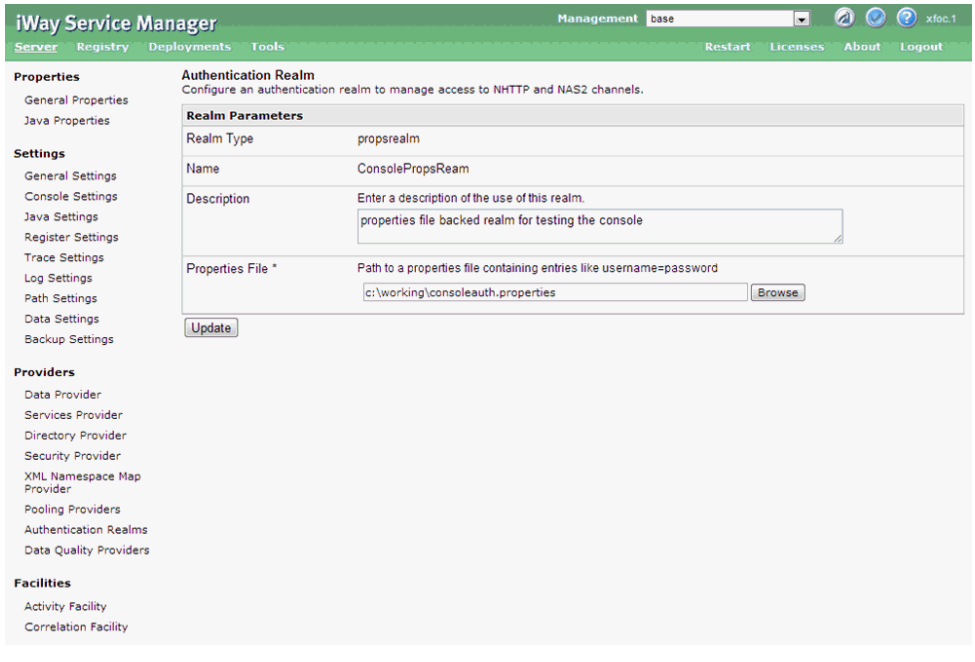
   This creates the following two users:

   - **Admin.** The iSM Administrator.
   - **User.** The assigned read-only role that was just created.

   The password for both users is *password*. For other types of realm, this step would be done differently, using whatever tools are appropriate for the system that backs the realm. For example, for an LDAP realm that uses Microsoft Active Directory, it would be necessary to work with user attributes in the directory.

5. In the iSM Administration Console, create the authentication realm to use the properties file by clicking on *Authentication Realms*, under Providers, of the Server menu.

6. Provide a name for the realm (for example, ConsolePropsRealm), its description, and the location of where the consoleauth.properties file is located, as shown in the following image.



7. Edit the server configuration so that the console will use the realm for authentication by clicking on *Management*, *Server Management*, and selecting *Servers*.

   The following table lists and describes the properties of the base configuration.

Parameter	Value	Description
Authentication Realm	ConsolePropsRealm	Realm that was just created.
Console Admin ID	Admin	The admin user in the realm.
Console Admin Password	password	The password assigned to the admin user.

8. Restart the base configuration and access the iSM Administration Console.

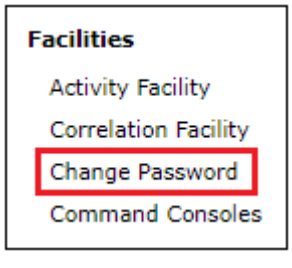
   When prompted, enter either *Admin* and *password* or *User* and *password* to log on to the console.

Managing User Accounts

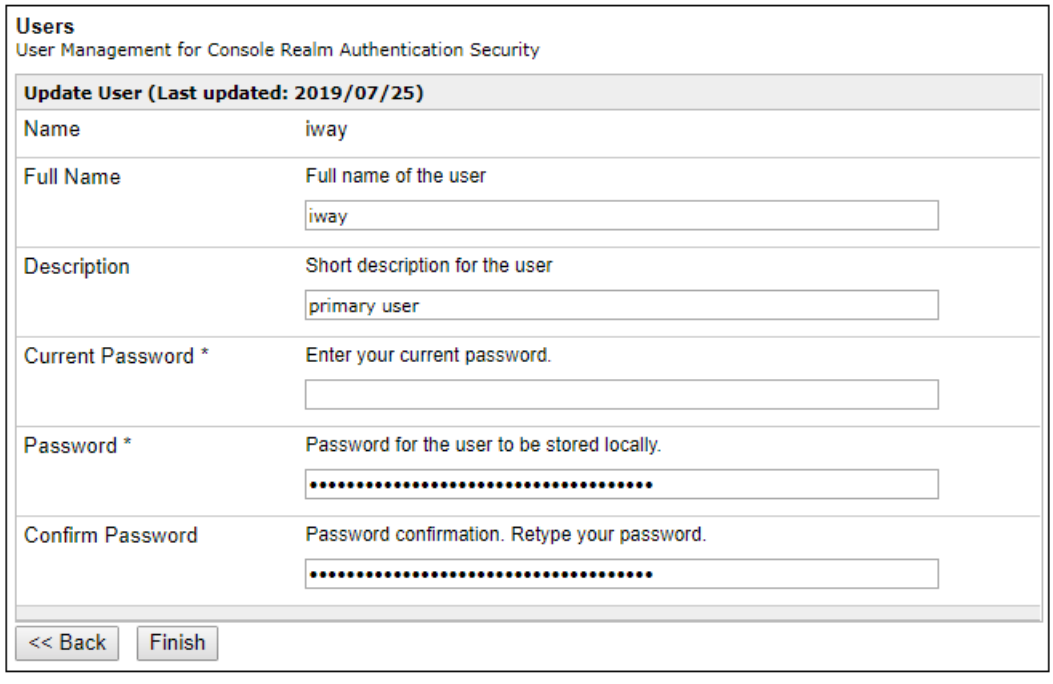
This section describes how to manage user accounts through the iWay Service Manager (iSM) Administration Console.

Changing Passwords

Users who are logged in to the iSM Administration Console can change their password using the Change Password facility. This facility applies only to users who are logged into the Console Realm (default security realm) and does not apply to other security realms, such as LDAP. To access this facility, click *Change Password* in the left pane under the Facilities section, as shown in the following image.



The Users page is displayed, which allows the logged in user to add to, or update their account information, as shown in the following image.

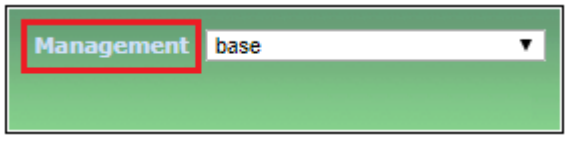


The following table lists and describes the available parameters the logged in user can modify.

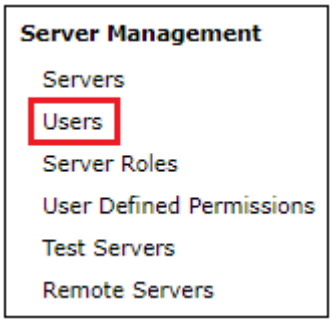
Parameter	Description
Full Name	Full name of the current user.
Description	A brief description for the user (for example, job title or department name).
Current Password	Specify a current password for authentication purposes.
Password	Specify a new password.
Confirm Password	Retype the new password for confirmation purposes.

Accessing Configured Users

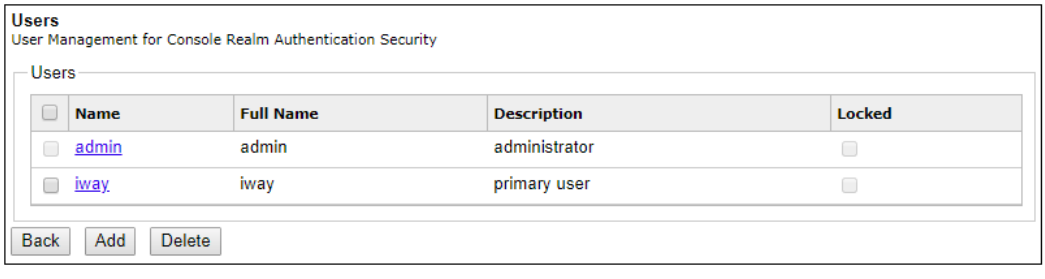
Administrators and non-administrative users that are allowed to reset passwords can access a list of currently configured iSM users by clicking *Management* from the menu bar in the iSM Administration Console, as shown in the following image.



Continue by selecting *Users* in the left pane under the Server Management section, as shown in the following image.



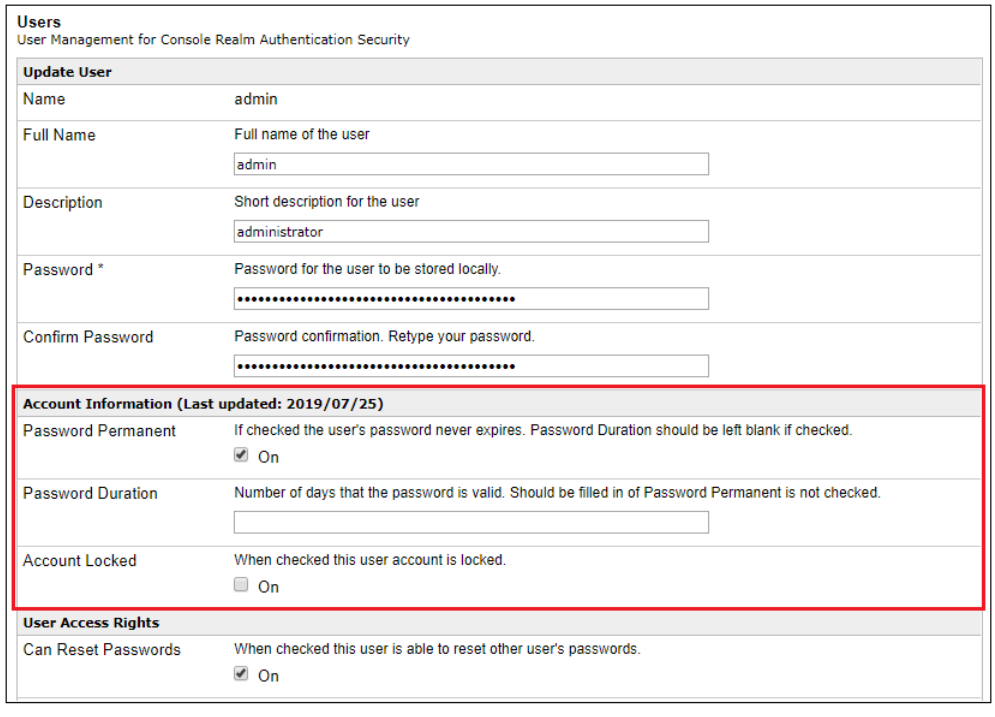
The Users page is displayed, as shown in the following image.



This Users page is available to administrators and users that are authorized to reset passwords. Note the Locked column that is available on this page. Accounts that are locked are prevented from accessing iSM functionality.

Password Control Parameters

Selecting an available user from the Users page displays a table containing configuration parameters for the user account. The Account Information section of the table provides fields to control passwords, as shown in the following image.

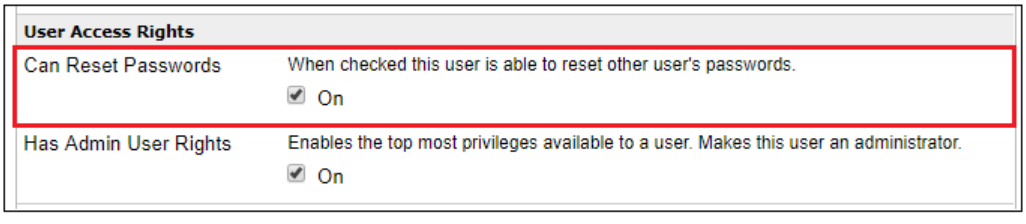


The following table lists and describes the available parameters in the Account Information section.

Parameter	Description
Password Permanent	When this check box is selected, the user's password never expires.

Parameter	Description
Password Duration	Specifies the number of days that the current password is valid. This field and the Password Permanent check box are mutually exclusive. If the Password Permanent check box is selected, then the Password Duration field should not contain a value.
Account Locked	When this check box is selected, the user's account is locked and is unable to access iSM.

Under the User Access Rights section, the Can Reset Passwords check box is available. This option allows the administrator to designate one or more individuals as users that have the capability to reset passwords.

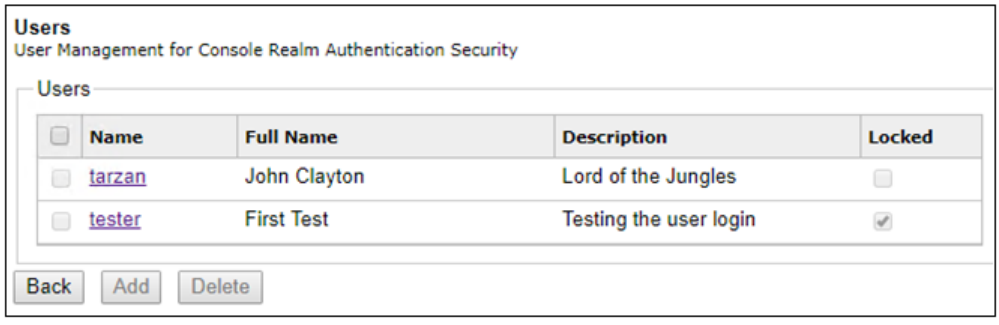


Users that are provided with this privilege can only reset another user's password and nothing else.

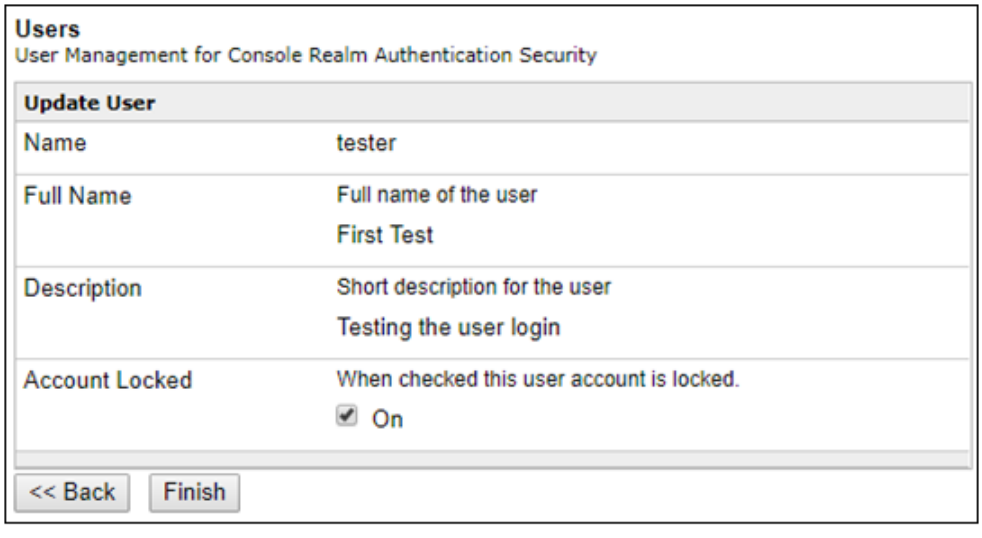
Non-Administrative Users

Users with the Can Reset Passwords privilege, will see a list of users that are locked out, as well as their own user ID when clicking *Management* from the menu bar in the iSM Administration Console, and then selecting *Users* in the left pane under the Server Management section.

In the following example, a sample user *tarzan* was granted with the Can Reset Passwords privilege.



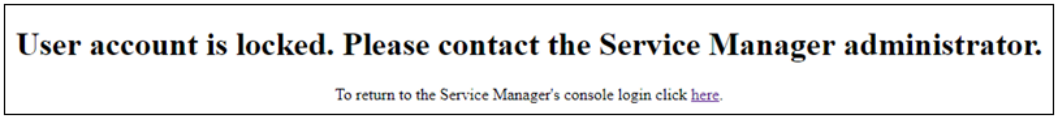
When selecting the *tester* user, the following screen is displayed.



Clearing the Account Locked check box will now allow the *tester* user to log on to iSM for the remainder of the current day. That user must change their password at some point during the current day. Otherwise, they will be locked out again on the following day.

Locked Accounts

If a user's account is locked by the administrator, or there were too many attempts to log in to iSM with invalid credentials, the user will see the following message displayed.



If a user enters their password incorrectly five times, their account will be temporarily locked for five minutes. After five minutes, the user can enter their password once again.

Chapter 6

Security Services

This section describes several security-related services, that can be used in a secure application. Not all security-related services are discussed here. For example, S/MIME services are documented in the *iWay Service Manager Component and Functional Language Reference Guide* along with many more general services available to applications.

In this chapter:

OAuth 1.0 Authentication Service

Syntax:

```
com.ibi.agents.XDOAuth1Agent
```

Description:

This service creates the HTTP Authorization header for OAuth 1.0a as specified in RFC5849. This RFC describes a 3-legged protocol where the user authorizes the client application to access a protected resource hosted by a service provider.

The OAuth 1.0 Authentication supports a variant called the 0-legged protocol where the request is signed without the user credentials. The signature is computed using just the consumer key and the consumer secret. These credentials are obtained once when the application programmer registers his client application with the service provider during development. This service assumes the consumer secret is a private key.

Parameters:

The following table lists and describes the parameters for the OAuth 1.0 Authentication service.

Parameter	Description
HTTPS URL	Request URL used in the computation of the Signature Base String.
HTTP Method	HTTP Method used in the computation of the Signature Base String. Selecting POST will also cause the current document to be hashed to produce the oauth_body_hash.
Header Namespace	Special register namespace where the Authorization HTTP header is stored. If not supplied, the default namespace will be used.
Client ID	The consumer key of the client credentials.
KeyStore Provider	Provider for the keystore containing the client private key.
Private Key Alias	Alias of the private key within the keystore.
Private Key Password	Password for the private key. If left blank, the password for accessing the keystore will be used.

The OAuth Authentication service only creates the Authorization header. The HTTP request must be sent in a separate step, usually with the NHTTP Emit service.

The HTTPS URL and HTTP Method parameters are used in the Signature Base String. They must match the Target URL and Action Method of the NHTTP Emit service. The URL scheme must be HTTPS because an SSL connection is needed to protect the information that is passed in clear. Choosing the POST method also instructs the service to compute a hash of the entity body to be part of the signature. This algorithm was specified by Google in its OAuth Request Body Hash extension.

The Authorization header will be stored in the specified Header Namespace. This parameter should match the Request Header Namespace in the NHTTP Emit Agent. This will ensure that the header is sent with the request. It is possible to use different namespaces, as long as the Authorization register is copied to the Request Header Namespace before the request is sent.

The Client ID is the consumer key supplied by the service provider when the developer registered the client application with the service provider. This serves as the user name for the client application. The service provider uses the client ID to retrieve the public key to validate the signature.

The KeyStore Provider is the name of the provider that holds the client private key. The Private Key Alias and Private Key Password are the Alias and Password for the private key. This key is used as the consumer secret when signing the Authorization header.

The output document is the same as the input document.

For the POST method, the document contains the same data but it will be stored as bytes if it was not already. This is to guarantee the document will not be altered before it is sent because any change to the document would invalidate the signature.

Edges:

The following table lists and describes the edges that are returned by the OAuth Authentication service.

Edge	Description
success	The Authorization header was successfully created.
fail_parse	An iFL expression could not be evaluated.
fail_operation	The operation could not be completed successfully.

Example 1

This example shows the creation of an OAuth 1.0a Authorization header for a GET method. The following table lists the parameter values for the service.

Parameter	Value
HTTPS URL	*https://sandbox.api.mastercard.com/atms /v1/atm?Format=XML&PageOffset=0&PageLength=10&PostalCode=46312&Country=USA*
HTTP Method	`GET`
Header Namespace	`hdrns`
Client ID	`DKB0vGSHs4r1Vv308yObMj4QhhJkIMP5G3a14KmEa7f96b5e!414a78536b4a6f6272634a41446e4566483851625a7a413d`
KeyStore Provider	`keyprov`
Private Key Alias	`key1`
Private Key Password	`key1pass`

This assumes key1 is the alias of a private key entry in the KeyStore provider keyprov. The service will compute the following Signature Base String. The oauth_nonce and oauth_timestamp will obviously change each time the service executes.

```
GET&https%3A%2F%2Fsandbox.api.mastercard.com%2Fatms%2Fv1%2Fatm&Country%3DUSA%26Format%3DXML%26PageLength%3D10%26PageOffset%3D0%26PostalCode%3D46312%26oauth_consumer_key%3DDKB0vGSHs4r1Vv308yObMj4QhhJkIMP5G3a14KmEa7f96b5e%2521414a78536b4a6f6272634a41446e4566483851625a7a413d%26oauth_nonce%3D180284899533025%26oauth_signature_method%3DRSA-SHA1%26oauth_timestamp%3D1396020436%26oauth_version%3D1.0
```

The service will store the following header value in the hdrns. Authorization special register. The oauth_signature changes every time the service is executed because the oauth_nonce and oauth_timestemp varies.

```
OAuth
oauth_signature="JjBI1gi5EMHwcihnCyK0RX7UzCC2SCtplutEjUgUXaI2nhGd4IR3L7b
WMtpJKkyUnR667lpkI7zqbM3oR3CHc2%2FgxPerD%2FSDGibHTAcTHCfV9%2F0xBVzv%2Fzo
1egU4CEqjZGSeIAeJKQYOflKSrfX8ken0MsXwXv5s9TLQuO8pRPwCfrqgrmVa%2FHhlzRxU7
pEv2kpJn4opG3Cvn01aKlotztxG8u476aEydFq03emqjVh8GMArtGDt8RhJqisJ0OB9SsaWU
K%2FsV%2BQtvghmX7G0pyQ6hLJUa3NSqlINU2k19cLOhUEnylDVD62sTZGrPe9%2B3zKLj%2
BX77eGLFKrDqOxk9w%3D%3D",oauth_version="1.0",oauth_nonce="18028489953302
5",oauth_signature_method="RSASHA1",oauth_consumer_key="DKB0vGSHs4r1Vv30
8yObMj4QhhJkIMP5G3a14KmEa7f96b5e%21414a78536b4a6f6272634a41446e456648385
1625a7a413d",oauth_timestamp="
1396020436"
```

If the Request Header Namespace is hdrns in the NHTTP Emit service, this will add the following HTTP header to the HTTP request.

```
Authorization: OAuth
oauth_signature="JjBI1gi5EMHwcihnCyK0RX7UzCC2SCtplutEjUgUXaI2nhGd4IR3L7b
WMtpJKkyUnR667lpkI7zqbM3oR3CHc2%2FgxPerD%2FSDGibHTAcTHCfV9%2F0xBVzv%2Fzo
1egU4CEqjZGSeIAeJKQYOflKSrfX8ken0MsXwXv5s9TLQuO8pRPwCfrqgrmVa%2FHhlzRxU7
pEv2kpJn4opG3Cvn01aKlotztxG8u476aEydFq03emqjVh8GMArtGDt8RhJqisJ0OB9SsaWU
K%2FsV%2BQtvghmX7G0pyQ6hLJUa3NSqlINU2k19cLOhUEnylDVD62sTZGrPe9%2B3zKLj%2
BX77eGLFKrDqOxk9w%3D%3D",oauth_version="1.0",oauth_nonce="18028489953302
5",oauth_signature_method="RSASHA1",oauth_consumer_key="DKB0vGSHs4r1Vv30
8yObMj4QhhJkIMP5G3a14KmEa7f96b5e%21414a78536b4a6f6272634a41446e456648385
1625a7a413d",oauth_timestamp="1396020436"
```

Example 2

This example shows the creation of an OAuth 1.0a Authorization header for a POST method. The following table lists the parameter values for the service.

Parameter	Value
HTTPS URL	*https://sandbox.api.mastercard.com/fraud/merchant/v1/termination-inquiry?Format=XML&PageLength=10&PageOffset=0*
HTTP Method	`POST`
Header Namespace	`hdrns`
Client ID	`DKB0vGSHs4r1Vv308yObMj4QhhJkIMP5G3 a14KmEa7f96b5e!414a78536b4a6f6272634a41446e4 566483851625a7a413d`
KeyStore Provider	`keyprov`

Parameter	Value
Private Key Alias	key1
Private Key Password	key1pass

The following input document is the parsed XML document:

```
<ns2:TerminationInquiryRequest xmlns:ns2="http://mastercard.com/
termination"><AcquirerId>1996
</AcquirerId><TransactionReferenceNumber>1</TransactionReferenceNumber>
<Merchant><Name>TEST</Name><DoingBusinessAsName>TEST
</DoingBusinessAsName><PhoneNumber>5555555555</PhoneNumber>
<NationalTaxId>1234567890</NationalTaxId><Address><Line1>5555 Test Lane
</Line1><City>TEST</City><CountrySubdivision>XX</CountrySubdivision>
<PostalCode>12345</PostalCode><Country>USA</Country></Address>
<Principal><FirstName>John</FirstName><LastName>Smith</LastName>
<NationalId>1234567890</NationalId><PhoneNumber>5555555555</PhoneNumber>
<Address><Line1>5555 TestLane</Line1><City>TEST</City><CountrySubdivision>XX
</CountrySubdivision><PostalCode>12345</PostalCode><Country>USA
</Country></Address><DriversLicense><Number>1234567890</Number>
<CountrySubdivision>XX</CountrySubdivision></DriversLicense></Principal>
</Merchant></ns2:TerminationInquiryRequest>
```

The service will compute the following Signature Base String. Notice the extra attribute oauth_body_hash compared to Example 1.

```
POST&https%3A%2F%2Fsandbox.api.mastercard.com%2Ffraud%
2Fmerchant%2Fv1%2Fterminationinquiry&Format%3DXML%26PageLength%3D10%
26PageOffset%3D0%26oauth_body_hash%3Dh3%252BhLMkT%252B3pBvRolKEc95fobEB8
%253D%26oauth_consumer_key%3DDKB0vGSHs4r1Vv308yObMj4QhhJkIMP5G3a14KmEa7f
96b5e%2521414a78536b4a6f6272634a41446e4566483851625a7a413d%26oauth_nonce
%3D180286176383600%26oauth_signature_method%3DRSA-SHA1%26oauth_timestamp
%3D1396020438%26oauth_version%3D1.0
```

The service will store the following header value in the hdrns Authorization special register. The oauth_signature, oauth_nonce, and oauth_timestamp will change every time the service is executed.

```
OAuth
oauth_signature="GSgJ6wUiYDznurpspn2ztn9PZeuXIBy4LZZHOSuMQrQ8OskwdWdaX0i
UXfNELxEQUniy6z5b2c06yVCut4XoYtV5XJaYnoG78bqkJ3LLVBqZ%2Brv%2F%2FTbIQmz0c
enMAinlR09QeduIHV7gPGqd%2FBi9Rkj%2BHnxI5bLNGn0nQoOie%2BSNUAPCjnn2Ydoj44l
Sufmur6N2U7paJAuEIfp3VANbLwCI%2Bts5EBr3ecCn7eEqbuQMzs8hW2c%2FdzZqoOvyEda
O86SVcTX9vT5XI8V%2FRluupobCRy8xSuxubnCJrf5USfT%2FB5rudqNkHW0%2BmtE8hxVLI
L9v2dKPSRxtqsU75GsrgA%3D%3D",oauth_body_hash="h3%2BhLMkT%2B3pBvRolKEc95f
obEB8%3D",oauth_version="1.0",oauth_nonce="180286176383600",oauth_signat
ure_method="RSASHA1",oauth_consumer_key="DKB0vGSHs4r1Vv308yObMj4QhhJkIMP
5G3a14KmEa7f96b5e%21414a78536b4a6f6272634a41446e4566483851625a7a413d",
oauth_timestamp="1396020438"
```

The output document is the same as the input but the data is now stored as bytes.

If the Request Header Namespace is hdrns in the NHTTP Emit service, this will add an Authorization header to the HTTP request.

OAuth 2.0 Authentication Service

Syntax:

```
com.ibi.agents.XDOAuth2Agent
```

Description:

This service emits an HTTPS request authenticated by OAuth 2.0 using the credentials of a Google service account.

OAuth 2.0 is described in RFC6749 and RFC6750. It is an authorization framework that enables an application to obtain access to an HTTP resource. The role of the client and the resource owner are separate. The client does not use the resource owner credentials. Instead, it requests an access token from a trusted authorization server. The client presents the token together with the request to the resource server which grants access if the token is valid.

The OAuth 2.0 Authentication service manages the creation and renewal of access tokens by communicating with the authorization server. If it obtains a valid access token, it incorporates the token with the outgoing HTTPS request to make the authenticated call.

When OAuth 2.0 is used interactively, the user is often redirected to a consent screen to enter his credentials. Since the iSM service operates in a server-to-server scenario, there is no consent screen involved. Instead, the client provides its credentials with the private key associated with a Google service account. The authorization server will be accessed at the same location as the resource server.

When obtaining a token, the client must specify the scope of the access requested. For example, there could be a scope for read-only and another for read-write permissions. It is also possible to request multiple scopes in a single access token.

The scopes are not standardized. The resource servers are free to define the scopes that make sense for their application. The documentation of the resource server API will make it clear which OAuth 2.0 scopes it supports. It remains the responsibility of the application designer to request and use the appropriate scopes.

Parameters:

The following table lists and describes the parameters for the OAuth 2.0 Authentication service.

Parameter	Description
HTTP Client Provider	HTTP Client provider that manages connections for this emitter.
Destination URL	URL where the request will be addressed. The URL should be fully specified, including the HTTPS scheme.
HTTP Method	POST sends the current document as a request entity. GET and HEAD will send a request to the configured URL.
Content Type	Content type for the HTTP request to be sent by this emitter.
Request Header Namespace	Special register namespace from which HTTP headers for the outgoing request will be taken. Select *Default Namespace* to send the HDR type registers with no namespace prefix, or supply a namespace prefix here. *None* means that no special registers will be sent as HTTP headers.
Response Header Namespace	Special register namespace into which HTTP headers from the incoming response will be saved. Select *Default Namespace* to create special registers with no namespace prefix, or supply a namespace prefix here.
Scopes	Determines which services the application requests access to. Select one from the list or enter a space-separated list of scopes.
Project ID	Value of the x-goog-project-id header.
Service Account Email	Email address of the service account.
KeyStore Provider	Provider for the keystore containing the service account private key.
Private Key Alias	Alias of the service account private key within the keystore.
Private Key Password	Password for the private key. If left blank, the password for accessing the keystore will be used.

The HTTP Client provider can be defined on the pooling providers page in the iWay Service Manager console. Google recommends that OAuth 2.0 should always be used with HTTPS, therefore the HTTP Client provider should specify an SSLContext Provider.

The authenticated request will be sent to the Destination URL. The service will access the authorization server at the same location to obtain access tokens. The HTTP method specifies the type of request: GET, PUT, POST, HEAD, PATCH or DELETE. The Content Type parameter specifies the content type of the authenticated request.

If additional headers are needed, they can be declared as HDR registers in the Request Header Namespace. The default value is *none*, which means there is no Request Header Namespace and no extra headers are added.

The headers of the HTTP response are stored as special registers in the specified Response Header Namespace.

The Scopes parameter specifies the access scope requested from the authorization server. The access token returned will grant those scopes. For example, if the scope in the token is read-only-access and a write operation is attempted, the resource server will reject the call because the scope is not sufficient. The scopes are application specific. For more information, see the resource server API documentation to learn which scopes it supports.

The Project ID is optional. When present, it specifies the value of the x-goog-project-id header. In the cloud platform of Google, a Project consists of a set of users, a set of APIs, and billing, authentication, and monitoring settings for those APIs.

The Service Account Email acts like a user name for a service account. It looks like an email address but there is no actual email involved. The credential for the service account is a private key. The KeyStore Provider is the name of the iSM keystore provider that holds the private key. The Private Key Alias and the Private Key Password are the alias and password of the private key entry within the Keystore.

Edges:

The following table describes the edges that are returned by the OAuth 2.0 Authentication Service.

Edge	Description
Success	The request message was successfully sent and the response received.
fail_parse	An iFL expression could not be evaluated.

Edge	Description
fail_operation	The operation could not be completed successfully.

Example:

This example shows how to retrieve a document from the Google Cloud Storage using the OAuth 2.0 Authentication Service.

Google Cloud Storage is a service to store data in the cloud and retrieve it later. It has a RESTful API, authenticated with OAuth 2.0. The equivalent of a file is called an object. Objects are stored in folders called buckets. Buckets are like directories, except they are not hierarchical. Every bucket exists in a single namespace shared by all users of the service. The slash (/) character is invalid in a bucket name but is accepted in an object name.

The Google Cloud platform supports multiple varieties of credentials. For server to server applications, a service account must be used. This is a name associated with a public key. The client proves his identity by encrypting with the private key which is kept secret. Service accounts are tied to a specific Google project. They are created in the Google Developer Console. Google chooses the service account name and creates the public-private key pair automatically. The private key is downloaded at the time the service account is created.

The Google Developer Console may change without notice. Proceed with the following steps:

From the following website, *https://console.developers.google.com*:

1. Sign in with your Google account or create a new google account on the sign-in page if you do not already have one. This is your regular Google account, not a service account yet.
2. Create a new Google project, specify the project name and the project ID, and then write down the project ID.
3. In the left menu, select *APIs & auth* and then click on *APIs*. Ensure that the Google Cloud Store is set to *ON*.
4. In the left menu, select *APIs & auth*, click *Consent*. and then proceed to the next step. The consent screen is not used in this example, but it must be selected and meet the requirements of Google.
5. Enter any Product Name and click *Save*.
6. In the left menu, select *APIs & auth*, click *Credentials*, and select *Create new Client ID*.
7. In the dialog that appears, select *Service Account* and click *Create client ID*. This will download a PKCS12 Keystore containing the private key of this service account.
8. Save the keystore in a convenient location and write down the keystore password shown on the screen, for example, notasecret.

9. Record the email address of the service account shown on the screen. The email address will look similar to the following:

   ```
   298643775104-81neakmsco3agrv956tl8inu8ci7oedl@developer.gserviceaccount.com
   ```

10. In the left menu, select *Billing & settings*, then enable the billing, and fill in the financial information.
11. In the left menu, click *Storage*, select *Cloud Storage*, *Storage Browser* and then click *Create a bucket*, and enter the bucket name. This must be a unique name among all buckets in the Google Cloud Storage. A good practice is to use your bucket name with your domain name changing the period with a dash. For example: test99-example-com
12. Click *Upload* and select a file to store in the Google Cloud Storage, then write down the name of the bucket and the file name in the bucket.
13. Go to *https://pki.google.com* and download the CA certificate for Google Internet Authority G2. This will be used to create a truststore.

    The following information will be available:

    - ❑ The project ID.
    - ❑ The Keystore containing the private key.
    - ❑ The Keystore password.
    - ❑ The service account email address.
    - ❑ The bucket name.
    - ❑ The object name in the bucket.
    - ❑ The certificate for Google Internet Authority G2.

14. Import the Google CA certificate into a Java Keystore with the following command. This assumes the certificate is stored in a file called GIAG2.cer

    ```
    keytool -import -trustcacerts -file GIAG2.cer -alias giag2 -keystore
    GIAG2.jks -storepass secret -storetype JKS
    ```

15. In the left menu of the iWay Service Manager console, click *Security Providers*, and then click *New* to create a new Keystore provider to be used as the truststore.

    The following table lists the parameters and values of the Keystore provider.

Parameter	Value
Name	giag2

Parameter	Value
Description	Google Internet Authority G2
Keystore	Path to GIAG2.jks
Keystore Password	secret
Keystore type	JKS
KeyStore JCE Provider	SUN

16. Click *New* again to create another Keystore provider for the private key of the service account.

Parameter	Description
Name	cloudkey
Keystore	Path to the keystore containing the service account private key.
Keystore Password	The keystore password chosen by Google, for example, notasecret.
Keystore type	PKCS12-DEF
KeyStore JCE Provider	BC

17. Create an SSLContext provider to be used by the HTTPS connections, and accept all default parameters except the following:

Parameter	Description
Name	GoogleSSL
Keystore Provider	Not used but required. You can enter giag2
Truststore Provider	giag2
Security Protocol	TLS

18. Create an HTTP Client provider to manage connections to the Google Cloud Storage by clicking *Pooling Providers* in the left menu of the iWay Service Manager console, and then clicking *New*.

19. Accept all default parameters except those found in the following table:

Parameter	Description
Name	GoogleClient
SSL Context Provider	GoogleSSL

20. In your process flow, create an instance of the OAuth 2.0 Authentication Service configured as shown in the following table. Replace the values in angle brackets with the actual value obtained in the Google Developers console.

Parameter	Description
HTTP Client Provider	GoogleClient
Destination URL	URL of the object in the Google Cloud Storage.
HTTP Method	GET
Scopes	*https://www.googleapis.com/auth/devstorage.read_only*
Project ID	The Google Project ID.
Service Account Email	The service account email address chosen by Google.
KeyStore Provider	cloudkey
Private Key Alias	privatekey
Private Key Password	Password chosen by google, for example, notasecret.

The Destination URL can take one of the following forms:

- *https://<bucketname>.storage.googleapis.com/<objectname>*
- *https://storage.googleapis.com/<bucketname>/<objectname>*

For example, if the bucket name is *mybucket-example-com*, and the object name is *root.xml*, the destination URL can be one of two equivalent URLs:

- *https://mybucket-example-com.storage.googleapis.com/root.xml*
- https://storage.googleapis.com/mybucket-example-com/root.xml

This instance of the OAuth 2.0 Authentication Service accepts any input document since the GET method does not send a request entity in the body of the message. The output document will be the contents of the object found in the Google Cloud Storage or an error document.

Insert WSSE Timestamp Service

Syntax:

```
com.ibi.agents.XDInsertWSSETimestampAgent
```

Description:

This service is used to generate a WSSE Timestamp.

Parameters:

The following table lists and describes the parameters for the Insert WSSE Timestamp service.

Parameter Name	Description
XML Namespace Provider	Provider for the mapping between XML namespace prefix and namespace URI. If left blank, the XPath expression that is specified for the Timestamp Parent Element parameter cannot contain namespaces.
XPath Version	Determines which implementation of XPath should be used. You can select the iWay implementation of XPath, an external XPath implementation, or the default. The default option selects the XPath implementation that is specified in the General Settings area of the iSM Administration Console.
Create Parent Element	Determines whether the parent element is created if it is missing. Select *true* or *false* (default) from the drop-down list.

Parameter Name	Description
Timestamp Parent Element	Path to the element where the timestamp will be inserted. The default value is: `/soapenv:Envelope/soapenv:Header/wsse:Security` If the Create Parent Element parameter is set to *true*, then the expression must adhere to Restricted XPath syntax, otherwise the expression may adhere to the full syntax of the XPath engine selected by the XPath Version parameter. Restricted XPath has the form `/step1/step2/...` where a step has the form `ns:elem[`*`predicate`*`]` or a pair of consecutive steps that has the form `*[1]/self::ns:elem[`*`predicate`*`]` to indicate the element must be the first child of its parent. The namespace prefixes are optional, but if present they must be declared in the XML Namespace provider. The predicate is optional, but when present it has the form `[@ns1:attr1='val1' and @ns2:attr2='val2' and ...]`. If no element matches the Restricted XPath expression and the Create Parent Element parameter is set to *true*, then the necessary elements and attributes will be created such that the expression would match successfully.
WSSE Timestamp Id	The value of the Timestamp ID attribute. Subsequent services can retrieve this value in the *wsse_timestamp_id* special register.
Expiration Period	The time period after which the timestamp will expire. If left blank, the timestamp will never expire. Use the following format: `[xxh][xxm]xx[s]` For example, 1m30s is 90 seconds.

Edges:

The following table lists and describes the edges that are returned by the Insert WSSE Timestamp service.

Edge	Description
success	The WSSE Timestamp was successfully inserted.
fail_parse	An iFL or XPath expression could not be evaluated.
fail_operation	The WSSE Timestamp could not be inserted.

The Created time is always the current time. The *Expires Time* is the current time plus the given Expiration Period. If the Expiration Period is left blank, the Expires element will not appear and the timestamp will never expire. The location where to insert the timestamp is given by an XPath expression that is specified in the Timestamp Parent Element parameter. The XPath expression can contain namespace prefixes if the optional XML Namespace Map Provider is specified. When the Create Parent Element parameter is true, the parent element will be created if needed, but the XPath expression must adhere to the Restricted XPath syntax. When the Create parent Element parameter is false, the parent element must exist but the expression may adhere to the full syntax of the XPath engine selected by the XPath Version parameter. The optional WSSE Timestamp Id parameter will generate a wsu:Id attribute if specified. The Id is saved in the wsse_timestamp_id special register. This can be used to refer to the Timestamp in an XML Digital Signature Reference using the URL expression #_sreg(wsse_timestamp_id).

The following example shows the creation of an WSSE Timestamp with an expiration period of 90 seconds. The following table lists the parameter values that were used.

Parameter Names	Value
XML Namespace Provider	
XPath Version	default
Create Parent Element	true
Timestamp Parent Element	
WSSE Timestamp Id	mytimestampid

Parameter Names	Value
Expiration Period	1m30s

A sample input document is shown as follows (indented for display purposes only):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>...</soapenv:Body>
</soapenv:Envelope>
```

The resulting document shows the addition of the Timestamp in the SOAP header block (indented for display purposes only):

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header>
  <wsse:Security xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <wsu:Timestamp wsu:Id="mytimestampid"
                  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-utility-1.0.xsd">
    <wsu:Created>2007-06-17T13:50:10.123Z</wsu:Created>
    <wsu:Expires>2007-06-17T13:51:40.123Z</wsu:Expires>
   </wsu:Timestamp>
  </wsse:Security>
 </soapenv:Header>
 <soapenv:Body>
  ...
 </soapenv:Body>
</soapenv:Envelope>
```

Insert WSSE Token Service

Syntax:

```
com.ibi.agents.XDInsertWSSETokenAgent
```

Description:

This service is used to generate a WSSE Binary Security Token containing an X509 certificate.

Parameters:

The following table lists and describes the parameters for the Insert WSSE Token service.

Parameter Name	Description
KeyStore Provider	Provider for the keystore containing the key.
Key Alias	Alias for the key to insert into the security token.
XML Namespace Provider	Provider for the mapping between XML namespace prefix and namespace URI. If left blank, elements in the security token will use the default namespace.
XPath Version	Determines which implementation of XPath should be used. You can select the iWay implementation of XPath, an external XPath implementation, or the default. The default option selects the XPath implementation that is specified in the General Settings area of the iSM Administration Console.
Create Parent Element	Determines whether the parent element is created if it is missing. Select *true* or *false* (default) from the drop-down list.

Parameter Name	Description
Security Token Parent Element	Path to the element where the security token will be inserted. The default value is: `/soapenv:Envelope/soapenv:Header/wsse:Security` If the Create Parent Element parameter is set to *true*, then the expression must adhere to Restricted XPath syntax, otherwise the expression may adhere to the full syntax of the XPath engine selected by the XPath Version parameter. Restricted XPath has the form `/step1/step2/...` where a step has the form `ns:elem[`*`predicate`*`]` or a pair of consecutive steps that has the form `*[1]/self::ns:elem[`*`predicate`*`]` to indicate the element must be the first child of its parent. The namespace prefixes are optional, but if present they must be declared in the XML Namespace provider. The predicate is optional, but when present it has the form `[@ns1:attr1='val1' and @ns2:attr2='val2' and ...]`. If no element matches the Restricted XPath expression and the Create Parent Element parameter is set to *true*, then the necessary elements and attributes will be created such that the expression would match successfully.
WSSE Security Token Id	The value of the BinarySecurityToken ID attribute. If left blank, the default value is x509_signer. Subsequent services can retrieve this value in the *wsse_token_id* special register.

Edges:

The following table lists and describes the edges that are returned by the Insert WSSE Token service.

Edge	Description
success	The BinarySecurityToken was successfully inserted.
fail_parse	An iFL or XPath expression could not be evaluated.
fail_operation	The BinarySecurityToken could not be inserted.

The WSSE Binary Security Token can later be referred to by an XML Digital Signature KeyInfo element and signed like any other XML content. The Keystore Provider and Key Alias specify which certificate will appear in the Security Token. There is no password to enter because you are only retrieving the public certificate corresponding to this private key. The location where to insert the Binary Security Token is given by an XPath expression specified in the Security Token Parent Element parameter. The XPath expression can contain namespace prefixes if the optional XML Namespace Map Provider is specified. When the Create Parent Element parameter is set to *true*, the parent element will be created if needed, but the XPath expression must adhere to the Restricted XPath syntax. When the Create Parent Element parameter is set to *false*, the parent element must exist but the expression may adhere to the full syntax of the XPath engine selected by the XPath Version parameter.

The optional WSSE Security Token Id parameter is used to generate a wsu:Id attribute on the wsse:BinarySecurityToken element. The Id is saved in the wsse_token_id special register. This can be used to refer to the security token in an XML Digital Signature Reference using the URL expression #_sreg(wsse_token_id). It can also be used to generate a KeyInfo/SecurityTokenReference with the Token Id expression _sreg(wsse_token_id).

The following example shows the creation of Binary Security Token. The following table lists the parameter values that were used.

Parameter Name	Value
KeyStore Provider	ksprov
Key Alias	alias1
XML Namespace Provider	
XPath Version	default

Parameter Name	Value
Create Parent Element	true
Security Token Parent Element	
WSSE Security Token Id	tokenid

This assumes there is a private key entry with alias alias1 in the keystore specified by the KeyStore provider ksprov.

A sample input document is shown as follows (indented for display purposes only):

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
<env:Body wsu:Id="bodyid"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">...</env:Body>
</env:Envelope>
```

The resulting output document shows the addition of the Binary Security Token in the SOAP header block (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <BinarySecurityToken EncodingType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" wsu:Id="tokenid" xmlns:wsu="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
MIICmDCCAgGgAwIBAgIBGDANBgkqhkiG9w0BAQUFADA6MQswCQYDVQQGEwJVUzEQMA4GA1UEChMH
...
</BinarySecurityToken>
      </wsse:Security>
   </env:Header>
   <env:Body wsu:Id="bodyid" xmlns:wsu="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">... </env:Body>
</env:Envelope>
```

Insert SAML Assertion Service

Syntax:

`com.ibi.agents.XDInsertSAMLAssertionAgent`

Description:

This service is used to generate a WSSE SecurityTokenReference containing an embedded SAML assertion.

Parameters:

The following table lists and describes the parameters for the Insert SAML Assertion service.

Parameter Name	Description
XML Namespace Provider	Provider for the mapping between XML namespace prefix and namespace URI. If left blank, elements in the security token will use the default namespace.
XPath Version	Determines which implementation of XPath should be used. You can select the iWay implementation of XPath, an external XPath implementation, or the default. The default option selects the XPath implementation that is specified in the General Settings area of the iSM Administration Console.
Create Parent Element	Determines whether the parent element is created if it is missing. Select *true* or *false* (default) from the drop-down list.

Parameter Name	Description
Security Token Parent Element	Path to the element where the security token reference will be inserted. The default value is: `/soapenv:Envelope/soapenv:Header/wsse:Security` If the Create Parent Element parameter is set to *true*, then the expression must adhere to Restricted XPath syntax, otherwise the expression may adhere to the full syntax of the XPath engine selected by the XPath Version parameter. Restricted XPath has the form `/step1/step2/...` where a step has the form `ns:elem[`*`predicate`*`]` or a pair of consecutive steps that has the form `*[1]/self::ns:elem[`*`predicate`*`]` to indicate the element must be the first child of its parent. The namespace prefixes are optional, but if present they must be declared in the XML Namespace provider. The predicate is optional, but when present it has the form `[@ns1:attr1='val1' and @ns2:attr2='val2' and ...]`. If no element matches the Restricted XPath expression and the Create Parent Element parameter is set to *true*, then the necessary elements and attributes will be created such that the expression would match successfully.
WSSE Security Token Reference Id	The value of the SecurityTokenReference ID Attribute. Subsequent services can retrieve this value in the *saml_token_id* special register.
SAML Assertion Id	The value of the SAML Assertion ID Attribute. Subsequent services can retrieve this value in the saml_assertion_id special register.
SAML Issue Instant	The value of the SAML IssueInstant attribute. Subsequent services can retrieve this value in the *saml_issue_instant* special register.
SAML Issuer	The value of the SAML Issuer attribute.

Parameter Name	Description
SAML Major Version	The value of the SAML MajorVersion attribute. The default value is 1.
SAML Minor Version	The value of the SAML MinorVersion attribute. The default value is 1.
SAML Authentication Instant	The value of the SAML AuthenticationInstant attribute.
SAML Authentication Method	The value of the SAML AuthenticationMethod attribute.
SAML Name Identifier Format	The value of the SAML NameIdentifier Format attribute.
SAML Name Identifier	The value of the SAML NameIdentifier element.
SAML Subject Confirmation Method	The value of the SAML ConfirmationMethod element.

Edges:

The following table lists and describes the edges that are returned by the Insert SAML Assertion service.

Edge	Description
success	The SAML assertion was successfully inserted.
fail_parse	An iFL or XPath expression could not be evaluated.
fail_operation	The SAML assertion could not be inserted.

The location where to insert the Security Token Reference is given by an XPath expression specified in the Security Token Parent Element. The XPath expression can contain namespace prefixes if the optional XML Namespace Map Provider is specified. When the Create Parent Element parameter is true, the parent element will be created if needed, but the XPath expression must adhere to the Restricted XPath syntax. When the Create parent Element parameter is false, the parent element must exist but the expression may adhere to the full syntax of the XPath engine selected by the XPath Version parameter. The optional WSSE Security Token Reference Id parameter is used to generate a wsu:Id attribute on the wsse:SecurityTokenReference element. The Id is saved in the saml_token_id special register. This can be used to refer to the security token in an XML Digital Signature Reference using the URL expression #_sreg(saml_token_id).

The required SAML Assertion Id is used to generate a saml:AssertionId attribute on the saml:Assertion element. The Assertion Id is saved in the saml_assertion_id special register for later reference. The required SAML Issue Instant is used to generate a saml:IssueInstant attribute on the saml:Assertion element. The issue instance is saved in the saml_issue_instant special register. As per the SAML schema, the following parameters are all required: SAML Issuer, SAML Major Version, SAML Minor Version, SAML Authentication Instant, SAML Authentication Method, SAML Name Identifier Format, SAML Name Identifier, and SAML Subject Confirmation Method. The Major and Minor Versions both default to 1.

The following sample shows a SAML Assertion created by the service. The following table lists the parameter values that were used.

Parameter Name	Value
XML Namespace Provider	
XPath Version	default
Create Parent Element	true
Security Token Parent Element	
WSSE Security Token Reference Id	tokenid
SAML Assertion Id	_a75adf55-01d7-40cc-929f-dbd8372ebdfc
SAML Issue Instant	2007-02-26T10:11:11Z
SAML Issuer	urn:tokenIssuer:sms:com
SAML Major Version	1

Parameter Name	Value
SAML Minor Version	1
SAML Authentication Instant	2007-02-26T10:11:00Z
SAML Authentication Method	urn:oasis:names:tc:SAML:1.0:am:X509-PKI
SAML Name Identifier Format	urn:oasis:names:tc:SAML:1.1:nameid-format:X509SubjectName
SAML Name Identifier	CN=Client gateway,O=Some client,C=GB
SAML Subject Confirmation Method	urn:oasis:names:tc:SAML:1.1:cm:sender-vouches

A sample input document is shown as follows (indented for display purposes only):

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>...</soapenv:Body>
</soapenv:Envelope>
```

The resulting document shows the addition of the WSSE SecurityTokenReference in the SOAP header block (indented for display purposes only):

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
 <soapenv:Header>
  <wsse:Security xmlns:wsse="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
   <SecurityTokenReference wsu:Id="mytokenid" xmlns:wsu="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd">
    <Embedded>
     <saml:Assertion saml:AssertionId="_a75adf55-01d7-40cc-929f-
dbd8372ebdfc" saml:IssueInstant="2007-02-26T10:11:11Z"
saml:Issuer="urn:tokenIssuer:sms:com" saml:MajorVersion="1"
saml:MinorVersion="1" xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
      <saml:AuthenticationStatement saml:AuthenticationInstant="2007-02-
26T10:11:00Z" saml:AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:X509-
PKI">
       <saml:Subject>
        <saml:NameIdentifier
saml:Format="urn:oasis:names:tc:SAML:1.1:nameid-
format:X509SubjectName">CN=Client gateway, O=Some client,
C=GB</saml:NameIdentifier>
        <saml:SubjectConfirmation>

<saml:ConfirmationMethod>urn:oasis:names:tc:SAML:1.1:cm:sender-
vouches</saml:ConfirmationMethod>
        </saml:SubjectConfirmation>
       </saml:Subject>
      </saml:AuthenticationStatement>
     </saml:Assertion>
    </Embedded>
   </SecurityTokenReference>
  </wsse:Security>
 </soapenv:Header>
 <soapenv:Body>
  ...
 </soapenv:Body>
</soapenv:Envelope>
```

XML Digital Signature Create Service

Syntax:

`com.ibi.agents.XDXMLDSigCreateAgent`

Description:

This service is used to generate an XML Digital Signature.

Parameters:

The following tables describe the parameters for the XML Digital Signature Create service. Each table is followed by a discussion of that parameter group.

Algorithms	
XML Digital Signature JCE Provider	JCE Provider for the XMLSignatureFactory service.
Canonicalization Method	Algorithm used to canonicalize the SignedInfo element before it is digested as part of the signature operation.
Canonicalization Method Parameters	Parameters for the Canonicalization Method. For Inclusive Canonical XML, this is empty. For Exclusive Canonical XML, this is a space-separated list of XML namespace prefixes.
Signature Method	Signature algorithm used to convert the canonicalized SignedInfo into the SignatureValue.

The XML Digital Signature JCE Provider selects the XML Digital Signature implementation. Choose *XMLDSig* for the implementation built-in to Java, or *ApacheXMLDSig* for the Apache Santuario implementation. ApacheXMLDSig is often a better choice because it is updated more frequently and therefore is more up to date.

The Canonicalization Method is the Algorithm used to canonicalize the SignedInfo element before it is digested as part of the signature operation. It can be the URI for Inclusive Canonical XML with or without comments, or the URI for Exclusive Canonical XML with or without comments. For Inclusive Canonical XML, the Canonicalization Method Parameters are empty. For Exclusive Canonical XML, the Canonicalization Method Parameters hold a space-separated list of XML namespace prefixes.

The Signature Method is the Signature algorithm used to convert the canonicalized SignedInfo into the SignatureValue. Notice the SHA1 algorithm is common, but is not considered secure anymore.

Signature Key	
KeyStore Provider	Provider for the keystore containing the signature private key.
Signing Key Alias	Private key alias used to sign the SignedInfo.
Signing Key Password	Password for the signing private key. If left blank, the password for accessing the keystore will be used.
Enforce KeyUsage Extension	If set to *true*, the verify certificates used for signing allow the digitalSignature KeyUsage extension.

The KeyStore Provider is the name of the provider that holds the private key. The Signing Key Alias and Signing Key Password are the Alias and Password for the private key. This key must be compatible with the signature algorithm chosen in the Signature Method parameter. When the Enforce KeyUsage Extension parameter is on, it will ensure certificates used for signing allow the digitalSignature KeyUsage extension.

Signature Location	
XML Namespace Provider	Provider for the mapping between XML namespace prefix and namespace URI. If left blank, elements in the XML Digital Signature will use the default namespace, and XPath expressions in the Parent and Next Sibling Paths cannot contain namespaces.
XPath Syntax	Determines which syntax level of XPath should be used. You can select the iWay abbreviated syntax or the XPath 1.0 full syntax. The default option selects the syntax level as set in the General Settings area of the iSM Administration Console.
Create Parent Element	Determines whether the parent element is created if it is missing. Select *true* or *false* (default) from the drop-down list.

Signature Location	
Signature Parent Element	Path to the element where the signature will be inserted. If left blank, then the signature parent is the root element. If the Create Parent Element parameter is set to *true*, then the expression must adhere to Restricted XPath syntax, otherwise the expression may adhere to the full syntax of the XPath engine selected by the XPath Syntax parameter. Restricted XPath has the form `/step1/step2/...` where a step has the form `ns:elem[`*`predicate`*`]` or a pair of consecutive steps that has the form `*[1]/self::ns:elem[`*`predicate`*`]` to indicate the element must be the first child of its parent. The namespace prefixes are optional, but if present they must be declared in the XML Namespace provider. The predicate is optional, but when present it has the form `[@ns1:attr1='val1' and @ns2:attr2='val2' and ...]`. If no element matches the Restricted XPath expression and the Create Parent Element parameter is set to *true*, then the necessary elements and attributes will be created such that the expression would match successfully.
Signature Next Sibling	Path to the next sibling node. The signature will be inserted before this node. If left blank, the signature is added as the last child of the parent.

The XML Namespace Provider is optional. It is the name of the provider that gives the mapping between XML Namespace prefixes and XML Namespace URIs. If left blank, the Signature Parent Element and Signature Next Sibling path expressions cannot contain namespace prefixes. The XML Namespace Provider is also used to choose a prefix for the Signature elements. If the http://www.w3.org/2000/09/xmldsig# namespace is not found, the generated Signature element will re-declare the default namespace to this URI.

The Signature Parent Element is an XPath expression pointing to the element where the ds:Signature element will be inserted.

The Signature Next Sibling is an XPath expression that points to a child of the parent element. The signature will be inserted before this node. If left blank, the signature is added as the last child of the parent.

When the Create Parent Element parameter is true, the parent element will be created if needed, but the XPath expression must adhere to the Restricted XPath syntax. When the Create parent Element parameter is false, the parent element must exist but the expression may adhere to the full syntax of the XPath engine selected by the XPath VersionXPath Syntax parameter.

Key Info	
Include Issuer Serial	Determines whether the X509IssuerSerial element is included in the KeyInfo X509Data element. Select *true* or *false* (default) from the drop-down list.
Include Subject Name	Determines whether the X509SubjectName element is included in the KeyInfo X509Data element. Select *true* (default) or *false* from the drop-down list.
Include Certificate Chain	Determines how much of the signer certificate chain is included in the KeyInfo X509Data element. Select one of the following values from the drop-down list: ❑ Complete Certificate Chain {complete} ❑ Signer Certificate Only {signer} (default) ❑ No Certificates {none}
Include WSSE Security Token Reference	Determines whether a WSSE SecurityTokenReference is included in the KeyInfo element. Select *true* or *false* (default) from the drop-down list.
WSSE Security Token Id	The value of the BinarySecurityToken ID attribute referenced by the WSSE SecurityTokenReference. If left blank, the default value is x509_signer.

These parameters determine the content of the generated KeyInfo element. They can be used alone or in any combinations. If none of the parameters are used, the KeyInfo element will not appear.

The Include Issuer Serial boolean parameter determines whether a KeyInfo/X509Data/X509IssuerSerial element is generated. This element uniquely describes the signer certificate by listing the Issuer DN and the certificate Serial Number.

The Include Subject Name boolean parameter determines whether a KeyInfo/X509Data/X509SubjectName element is generated. This element contains the signer certificate subject DN.

The Include Certificate Chain parameter determines how many certificates in the certificate chain are included in the KeyInfo. The choices are: no certificates, just the signer certificate, or all certificates. Each certificate is base64 encoded in a separate KeyInfo/X509Data/X509Certificate element.

The Include WSSE Security Token Reference parameter determines whether a KeyInfo/SecurityTokenReference element is generated to point to a previously generated WSSE Binary Security Token. If on, the WSSE Security Token Id parameter specifies the Id of the existing Binary Security Token. The InsertWSSETokenAgent is a convenient way to pre generate the Binary Security Token. In that case, the Security Token Id can be retrieved with the expression SREG(wsse_token_id).

ID Attributes	
Signature Id	The value of the Signature Id Attribute. If left blank, the generated Signature element will not have an Id attribute.
SignatureValue Id	The value of the SignatureValue Id Attribute. If left blank, the generated SignatureValue element will not have an Id attribute.
SignedInfo Id	The value of the SignedInfo Id Attribute. If left blank, the generated SignedInfo element will not have an Id attribute.
KeyInfo Id	The value of the KeyInfo Id Attribute. If left blank, the generated KeyInfo element will not have an Id attribute.

ID Attributes	

ID Attributes	Space-separated list of attributes that are considered type ID. The value of an ID attribute can be used in a same-document reference with a URI of the form #idvalue. Each attribute declaration has the form
	`ns:*/@ns1:attrib`
	or
	`@ns1:attrib`
	In this declaration, ns: and ns1: are optional. If used, the ns and ns1 prefixes must be declared in the XML Namespace Provider parameter.
	The form @ns1:attrib means an attribute named attrib in XML Namespace ns1. The form ns:*/@ns1:attrib is similar except the attribute must also appear on an element of any name in the XML Namespace ns. The default value is:
	`xml:id ds:*/@Id wsu:Id xenc:Id`

The Signature Id, SignedInfo Id, SignatureValue Id and KeyInfo Id parameters specify the value of the Id attribute on the Signature, Signature/SignedInfo, Signature/SignatureValue and Signature/KeyInfo elements respectively. If left blank, the Id attribute will not appear.

The ID Attributes parameter is a space-separated list of attributes that are considered type ID. The value of an attribute can be used in a same-document reference with a URI of the form #idvalue but only if it is declared of type ID. This parameter performs this type assignment. Each attribute declaration has the form ns:*/@ns1:attrib or @ns1:attrib where ns: and ns1: are optional. If used, the ns and ns1 prefixes must be declared in the XML Namespace Provider. The form @ns1:attrib means an Attribute named attrib in XML Namespace ns1. The form ns:*/@ns1:attrib is similar except the attribute must also appear on an element of any name in the XML Namespace ns. The default value is:

```
xml:id ds:*/@Id wsu:Id xenc:Id
```

The namespace prefix actually used is not important. Only the namespace URI is used to find a match.

Reference 1	
Reference 1 URI	URI to the first piece of data that will be digested and signed. If left blank, then the whole XML document will be digested and signed.
Reference 1 Digest Method	Digest algorithm applied to the first reference data (after Transforms are applied if specified) to yield the DigestValue.
Reference 1 Transform 1	First transform algorithm to apply to the first reference data.
Reference 1 Transform 1 Parameters	Parameters for the first transform algorithm to apply to the first reference data. For Exclusive Canonical XML, this is a space-separated list of XML namespace prefixes. For XSLT, this is the name of a defined transform. For XPathFilter, this is an XPath expression. For XPathFilter2, this is the string intersect, subtract or union, followed by an XPath expression. For more XPathFilter2 XPathType clauses, create user parameters called ref1transform1parms[Z], ref1transform1parms[Z]nsmap where Z >= 2. For STR-Transform, this is the canonical method URI.
Reference 1 Transform 1 XML Namespace Provider	Provider for the XML Namespace Map for XPathFilter and XPathFilter2 transforms.
Reference 1 Transform 2	Second transform algorithm to apply to the first reference data.
Reference 1 Transform 2 Parameters	Parameters for the second transform algorithm to apply to the first reference data. For Exclusive Canonical XML, this is a space-separated list of XML namespace prefixes. For XSLT, this is the name of a defined transform. For XPathFilter, this is an XPath expression. For XPathFilter2, this is the string intersect, subtract or union, followed by an XPath expression. For more XPathFilter2 XPathType clauses, create user parameters called ref1transform2parms[Z], ref1transform2parms[Z]nsmap where Z >= 2. For STR-Transform, this is the canonical method URI.

Reference 1	
Reference 1 Transform 2 XML Namespace Provider	Provider for the XML Namespace Map for XPathFilter and XPathFilter2 transforms.

The reference URIs supported are: <empty string> for the whole XML document; #idattrib for the same-document sub-tree rooted at the element that has an ID attribute with value idattrib; cid:contentid for the attachment that has a Content-ID header with value <contentid>; http://host:port/page for the resource located at this HTTP address, and possibly other URLs built-in to Java.

The Reference 1 URI parameter is the URI to the first piece of data that will be digested and signed. If left blank, then the whole XML document will be digested and signed.

The Reference 1 Digest Method is the digest algorithm applied to the reference data (after Transforms are applied if specified) to yield the DigestValue. Example choices are the full URI corresponding to sha1, sha256, or sha512 plus some others.

The Reference 1 Transform 1 is the first transform algorithm to apply to the reference data. The Reference 1 Transform 1 Parameters contain the parameters for the transform.

The Reference 1 Transform 2 is the second transform and Reference 1 Transform 2 Parameters specify its parameters.

Reference 2	
Reference 2 URI	URI to the second piece of data that will be digested and signed. If you need more references, create user parameters named ref[X]uri, ref[X]digest, ref[X]transform[Y], ref[X]transform[Y]parms[Z] where X >= 3, Y >= 1, Z >= 1. For example, ref3transform2 is the second transform of the third reference.
Reference 2 Digest Method	Digest algorithm applied to the second reference data (after Transforms are applied if specified) to yield the DigestValue.
Reference 2 Transform 1	First transform algorithm to apply to the first reference data.

Reference 2	
Reference 2 Transform 1 Parameters	Parameters for the first transform algorithm to apply to the second reference data. For Exclusive Canonical XML, this is a space-separated list of XML namespace prefixes. For XSLT, this is the name of a defined transform. For XPathFilter, this is an XPath expression. For XPathFilter2, this is the string intersect, subtract or union, followed by an XPath expression. For more XPathFilter2 XPathType clauses, create user parameters called ref2transform1parms[Z], ref2transform1parms[Z]nsmap where Z >= 2. For STR-Transform, this is the canonical method URI.
Reference 2 Transform 1 XML Namespace Provider	Provider for the XML Namespace Map for XPathFilter and XPathFilter2 transforms.
Reference 2 Transform 2	Second transform algorithm to apply to the second reference data.
Reference 2 Transform 2 Parameters	Parameters for the second transform algorithm to apply to the second reference data. For Exclusive Canonical XML, this is a space-separated list of XML namespace prefixes. For XSLT, this is the name of a defined transform. For XPathFilter, this is an XPath expression. For XPathFilter2, this is the string intersect, subtract or union, followed by an XPath expression. For more XPathFilter2 XPathType clauses, create user parameters called ref2transform1parms[Z], ref2transform2parms[Z]nsmap where Z >= 2. For STR-Transform, this is the canonical method URI.
Reference 2 Transform 2 XML Namespace Provider	Provider for the XML Namespace Map for XPathFilter and XPathFilter2 transforms.

Subsequent references 2, 3, ... are similar to reference 1 except a missing reference URI indicates the end of the list of references instead of the whole document.

The list of transforms per reference is not limited to 2. Any number of transforms can be specified using user parameters.

The list of references is not limited to 2. Any number of references can be specified using user parameters.

The following table lists the transforms available. Some transforms have implicit parameters and do not require any explicit parameters. Other transforms take parameters as described in the table.

Transforms Available to Digital Signature Service	
Attachment Complete Signature Transform http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Complete-Signature-Transform	This transform indicates that both the content and selected headers of the MIME part are referenced for signing. The following MIME headers are to be included(when present): ❑ Content-Description ❑ Content-Disposition ❑ Content-ID ❑ Content-Location ❑ Content-Type This transform takes no explicit parameters.
Attachment Content Signature Transform *http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform*	This transform indicates that only the content of the MIME part is referenced for signing. All of the MIME headers associated with the MIME part are ignored and not included in the output octet stream. This transform takes no explicit parameters
Base64 *http://www.w3.org/2000/09/xmldsig#base64*	This transform decodes the Base64 encoded character data. If the input is a node-set, then the string-value of the node-set is decoded (ignoring the element tags, comments and processing instructions). This transform takes no explicit parameters.
Enveloped Signature *http://www.w3.org/2000/09/xmldsig#enveloped-signature*	This transform removes the Signature element from the calculation of the signature when the signature is within the content that it is being signed. This transform takes no explicit parameters.

Transforms Available to Digital Signature Service	
Exclusive Canonical XML *http://www.w3.org/2001/10/xml-exc-c14n#*	This transform is useful when message parts can be enveloped and stripped off to construct new messages. Exclusive Canonical XML ignores the namespace context inherited from parent elements. This keeps the digested data constant despite these operations. This transform takes an optional space-separated list of XML namespace prefixes declared in the XML Namespace provider. These are additional prefixes to be ignored.
Exclusive Canonical XML With Comments *http://www.w3.org/2001/10/xml-exc-c14n#WithComments*	This transform is similar to Exclusive Canonical XML except comments are preserved in the digested data. This transform takes an optional space-separated list of XML namespace prefixes declared in the XML Namespace provider. These are additional prefixes to be ignored.
XPathFilter *http://www.w3.org/TR/1999/REC-xpath-19991116*	This transform evaluates the XPath expression for each node in the input node-set and keeps only the nodes where the expression evaluated to true. This transform takes the XPath expression in ref[X]transform[Y]parms1 and optionally an XML Namespace provider name in ref[X]transform[Y]parms1nsmap to declare a namespace map.
XPathFilter2 *http://www.w3.org/2002/06/ xmldsig-filter2*	This transform computes a filter node-set. The output is the intersection of the input node-set and the filter node-set. The filter node-set is computed by evaluating a sequence of XPath expressions and combining their result by intersection, subtraction or union. Each XPathType can be declared by a pair of parameters: ref[X]transform[Y]parms[Z] and optionally ref[X]transform[Y]parms[Z]nsmap. ref[X]transform[Y]parms[Z] must start with the string intersect, subtract or union, followed by an XPath expression. ref[X]transform[Y]parms[Z]nsmap if present must be the name of an XML Namespace Provider to declare the Namespace map for this XPathType.

Transforms Available to Digital Signature Service	
XSLTTransform *http://www.w3.org/TR/1999/REC-xslt-19991116*	This transform indicates an XSLT stylesheet must be used and the result is what is referenced for signing. This transform takes the name of a defined transform as parameter (similar to what is done with the XDGenTransform service). The defined transform must be an XSLT transform and return XML.
Inclusive Canonical XML *http://www.w3.org/TR/2001/REC-xml-c14n-20010315*	This transform performs typical XML Canonicalization that attracts the xml namespace declarations from the inherited context. This canonicalization is the default if the last transform returns a node-set. This transform takes no parameters.
Inclusive Canonical XML With Comments *http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments*	This transform is similar to Inclusive Canonical XML except comments are preserved. This transform takes no parameters.
Security Token Reference (STR) Transform *http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform*	This transform is used to resolve security token references, replacing the reference by the actual token in the signature. This transform takes an optional parameter for the canonical method URI in ref[X]transform[Y]parms1. The default is the Exclusive Canonical XML method http://www.w3.org/2001/10/xml-exc-c14n#. If the canonical method selected is the Exclusive Canonical XML method (with or without comments), then the transform also takes an optional space-separated list of XML namespace prefixes in the user parameter ref[X]transform[Y]parms1parms1. These are additional prefixes to be ignored.

If you need more references, create user parameters named ref[X]uri, ref[X]digest, ref[X]transform[Y], ref[X]transform[Y]parms[Z] where X >= 3, Y >= 1 and Z >= 1. For example, ref3transform2 is the second transform of the third reference.

When the Tree level is selected in the trace settings, the service will log the referenced data that was actually digested.

Edges:

The following table lists and describes the edges that are returned by the XML Digital Signature Create service.

Edge	Description
success	The Signature was successfully inserted.
fail_parse	An iFL or XPath expression could not be evaluated.
fail_operation	The Signature could not be inserted.

Examples

The examples in this section are specific to the XML Digital Signature Create service (com.ibi.agents.XDXMLDSigCreateAgent). For your convenience, the sample input and output documents are attached to the PDF in unabbreviated and unindented form.

For PDF-compatibility purposes, the file extension of the *XMLDSigCreate.zip* file is temporarily renamed to *.zap*. After saving this file to your file system, you must rename this extension back to *.zip* before the file can be used.

Example 1: Enveloped Signature

The following example is a very simple Signature. There is only one reference that englobes the whole document. The signing key is the private key entry at alias alias1 within the keystore specified by the KeyStore Provider ksprov. The KeyInfo mentions the Subject Name to help the verifier retrieve the signer certificate from a certificate store. The Subject Name is convenient for users but the Issuer Name and Serial Number would have made the certificate retrieval less ambiguous. Notice the use of the Signature Next Sibling parameter to insert the Signature before the current first child of the root element. By default, the Signature element would appear as the last child of the root element instead. Since the default empty reference URI is used to sign the whole document, you must also specify the Enveloped Signature transform to avoid signing the Signature element.

This table lists the parameter values for this example. Other parameters that are not listed have their default value.

Parameter	Value
XML Digital Signature JCE Provider	XMLDSig
Canonicalization Method	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Signature Method	http://www.w3.org/2000/09/xmldsig#rsa-sha1
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
Signature Next Sibling	/*/*[1]
Include Subject Name	true
Include Certificate Chain	No Certificates
Reference 1 Transform 1	http://www.w3.org/2000/09/xmldsig#enveloped-signature

A sample input document is shown as follows (indented for display purposes only):

```
<Message>
  <Body>...</Body>
</Message>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
         <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <Reference URI="">
            <Transforms>
               <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>jr1PaanqeXi4Gluc8FaNcN/lauc=</DigestValue>
         </Reference>
      </SignedInfo>
      <SignatureValue>
h0zzL340inx0GXcYvYHAUTdxBe0YgIDQgr4AVgi+JkPK4RSZF8dk+8VIeCOh1MhVzAlp2rDP7C5dsXZyjAFoMMvHGhSUQI1v
HiPR38Uu+we6THH+A+65i3mn9pCHPrJ8D9cdbegIlej8cyejT0DtiE5QT7qepO96xLxDnFpyJvU=</SignatureValue>
      <KeyInfo>
         <X509Data>
            <X509SubjectName>CN=Example Signer,O=iWay,C=US</X509SubjectName>
         </X509Data>
      </KeyInfo>
   </Signature>
   <Body>...</Body>
</Message>
```

Example 2: Simple SOAP Message

The following example shows how to sign a SOAP message by inserting a digital signature in the SOAP Header. The WSSE Security profile suggests to create separate References for SOAP header blocks and the SOAP Body. Since our sample input document does not have a SOAP Header, a single reference to the SOAP Body is all that is required. Since the Body is a fragment of the whole document, it is recommended to apply an Exclusive Canonical XML transform. The reference URL has the form #id, where the id is the value of an ID attribute on the referenced node. By default, the service recognizes the following ID attributes:

```
xml:id ds:*/@Id wsu:Id xenc:Id
```

In the sample input document below, the ID attribute on the SOAP Body is wsu:Id. Since this ID attribute is recognized by default, you do not need to declare a new ID attribute in the ID Attributes parameter. The path to the parent element of the Signature contains elements in namespaces, so an XML Namespace provider is necessary to define the prefixes used in the XPath expression. The SOAP Header element will be created automatically because the service is instructed to create the parent element. The syntax `*[1]/self::soapenv:Header` instructs the Restricted XPath engine to create the Header element as the first child of the SOAP Envelope before the Body element. The signing key is the private key entry at alias alias1 within the keystore specified by the KeyStore Provider ksprov. The KeyInfo contains the Issuer Name and Serial Number to unambiguously specify the signing certificate. The verifier will need to retrieve that certificate from a certificate store to verify the Signature.

It is assumed the XML Namespace provider xmlnsprov defines these prefixes:

Prefix	XML Namespace
soapenv	http://schemas.xmlsoap.org/soap/envelope/
wsse	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd

This table lists the parameter values for this example. Other parameters that are not listed have their default value.

Parameter	Value
XML Digital Signature JCE Provider	XMLDSig
Canonicalization Method	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Signature Method	http://www.w3.org/2000/09/xmldsig#rsa-sha1
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
XML Namespace Provider	xmlnsprov
Create Parent Element	true

Parameter	Value
Signature Parent Element	/soapenv:Envelope/*[1]/self::soapenv:Header/ wsse:Security
Include Issuer Serial	true
Include Subject Name	false
Include Certificate Chain	No Certificates
Reference 1 URI	#bodyid
Reference 1 Digest Method	http://www.w3.org/2000/09/xmldsig#sha1
Reference 1 Transform 1	http://www.w3.org/2001/10/xml-exc-c14n#

A sample input document is shown as follows (indented for display purposes only):

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Body wsu:Id="bodyid" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">...</env:Body>
</env:Envelope>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
               <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
               <ds:Reference URI="#bodyid">
                  <ds:Transforms>
                     <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                  </ds:Transforms>
                  <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <ds:DigestValue>uoCjbXYPSRb4PAGQ0wZpmQ2PAFY=</ds:DigestValue>
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>
omh50ZhF+MoR7IfaLoWYkEoFRrz9RCJmLkwLfqTx7KyIkEX7I3gOH9Loc+mbj8G1GPMAD5CrL4rRoegkFeK8M+2um/K6kdx3
c+bdcbPGMNfkg1Jive0KUbR0RPEJ1m0dUYWsoYNPeyrL7FzsydofwUCdF3XPfmomVSqGNoV1+eQ=</ds:SignatureValue>
            <ds:KeyInfo>
               <ds:X509Data>
                  <ds:X509IssuerSerial>
                     <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                     <ds:X509SerialNumber>16</ds:X509SerialNumber>
                  </ds:X509IssuerSerial>
               </ds:X509Data>
            </ds:KeyInfo>
         </ds:Signature>
      </wsse:Security>
   </env:Header>
   <env:Body wsu:Id="bodyid" xmlns:wsu="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">...</env:Body>
</env:Envelope>
```

Example 3: WSSE SecurityTokenReference

The following is an example of a signature over the SOAP Body with a WSSE SecurityTokenReference in the KeyInfo pointing to a WSSE BinarySecurityToken previously generated in the SOAP Header. The security token and the digital signature appear in the same wsse:Security Header block. The signature has two references, one for the BinarySecurityToken in the header, and another for the SOAP Body.

The XDInsertWSSETokenAgent can create the BinarySecurityToken in the SOAP Header. For more information about the XDInsertWSSETokenAgent and an example, see *Insert WSSE Token Service* on page 87. The output of the XDInsertWSSETokenAgent example will be used as the input document of this digital signature example.

The following table lists the parameter values of the XDXMLDSigCreateAgent for this example.

Parameter	Value
XML Digital Signature JCE Provider	XMLDSig
Canonicalization Method	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Signature Method	http://www.w3.org/2000/09/xmldsig#rsa-sha1
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
XML Namespace Provider	xmlnsprov
Create Parent Element	false
Signature Parent Element	/soapenv:Envelope/soapenv:Header/wsse:Security
Include Subject Name	false
Include Certificate Chain	No Certificates
Include WSSE Security Token Reference	true
WSSE Security Token Id	tokenid
Reference 1 URI	#tokenid
Reference 1 Digest Method	http://www.w3.org/2000/09/xmldsig#sha1
Reference 1 Transform 1	http://www.w3.org/2001/10/xml-exc-c14n#
Reference 2 URI	#bodyid
Reference 2 Digest Method	http://www.w3.org/2000/09/xmldsig#sha1
Reference 2 Transform 1	http://www.w3.org/2001/10/xml-exc-c14n#

The input document is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <BinarySecurityToken EncodingType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" wsu:Id="tokenid" xmlns:wsu="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
1.0.xsd">MIICmDCCAgGgAwIBAgIBGDANBgkqhkiG9w0BAQUFADA6MQswCQYDVQQGEwJVUzEQMA4GA1UEChMH
</BinarySecurityToken>
      </wsse:Security>
   </env:Header>
   <env:Body wsu:Id="bodyid" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">...</env:Body>
</env:Envelope>
```

The output of the XMLDSigCreateAgent is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <BinarySecurityToken EncodingType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" wsu:Id="tokenid" xmlns:wsu="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-
secext-
1.0.xsd">MIICmDCCAgGgAwIBAgIBGDANBgkqhkiG9w0BAQUFADA6MQswCQYDVQQGEwJVUzEQMA4GA1UEChMH
</BinarySecurityToken>
         <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
            <SignedInfo>
               <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
               <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
               <Reference URI="#tokenid">
                  <Transforms>
                     <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  </Transforms>
                  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <DigestValue>duT/tcXl+1fbuxyXZHeEBrMy+Fs=</DigestValue>
               </Reference>
               <Reference URI="#bodyid">
                  <Transforms>
                     <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  </Transforms>
                  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <DigestValue>XLnuUJo+oiZ/KnAfleCuMqNfRfM=</DigestValue>
               </Reference>
            </SignedInfo>
            <SignatureValue>gpUiN0PA/YEIYhXVaQcdkacKPCdBBB28nH3Yf6IpMnFmvJkkYAVXAhPLEkR48otTAtfnOIK2go0V
whnniD0SXUg53WNWfhxp4xUWvZASreygVLXLs83I5NnsR0WQKZxRR/9T3Qo7DzsOetDWmHyTehj
bVMJMKwWRSGHk+N9DI=</SignatureValue>
         </KeyInfo>
      </Signature>
   </wsse:Security>
  </env:Header>
  <env:Body wsu:Id="bodyid" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">...</env:Body>
</env:Envelope>
```

Example 4: Security Token Reference Transform

This example shows how a SecurityTokenReference can be signed with the Security Token Reference (STR) Transform. The parameter of the STR-Transform is a canonicalization method. The example selects the Exclusive Canonical XML, which itself requires a parameter for the additional namespace prefixes to ignore. Since the parameter of a transform parameter is rare, the value must be entered as a user parameter based on the following naming convention:

```
ref[X]transform[Y]parms1parms1
```

The signature has two references. The first reference covers the SOAP Body. The second reference covers the KeyInfo element containing the SecurityTokenReference. Signing a token reference is not secure, which is why the STR-Transform is used to resolve the token reference. This replaces the reference by the actual token in the signature. The STR-Transform is the first transform (Y=1) of the second reference (X=2). Therefore the name of the user parameter for the namespace prefixes is:

```
ref2transform1parms1parms1
```

The XDInsertWSSETokenAgent can create the BinarySecurityToken in the SOAP Header. For more information about the XDInsertWSSETokenAgent and an example, see *Insert WSSE Token Service* on page 87. The output of the XDInsertWSSETokenAgent example will be used as the input document of this digital signature example.

The following table lists the parameter values of the XDXMLDSigCreateAgent that are used for this example.

Parameter	Value
XML Digital Signature JCE Provider	XMLDSig
Canonicalization Method	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Signature Method	http://www.w3.org/2000/09/xmldsig#rsa-sha1
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
XML Namespace Provider	xmlnsprov
Create Parent Element	false
Signature Parent Element	/soapenv:Envelope/soapenv:Header/wsse:Security
Include Subject Name	false
Include Certificate Chain	No Certificates
Include WSSE Security Token Reference	true

Parameter	Value
WSSE Security Token Id	tokenid
KeyInfo Id	keyinfoid
Reference 1 URI	#bodyid
Reference 1 Digest Method	http://www.w3.org/2000/09/xmldsig#sha1
Reference 1 Transform 1	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Reference 2 URI	#tokenid
Reference 2 Digest Method	http://www.w3.org/2000/09/xmldsig#sha1
Reference 2 Transform 1	http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#STR-Transform
Reference 2 Transform 1 Parameters	http://www.w3.org/2001/10/xml-exc-c14n#

The PrefixList of the Exclusive Canonical XML method of the STR-Transform is specified with this user parameter. The parameter name and the value must be entered as shown in the following table. The namespace prefix list is space separated.

Parameter	Value
ref2transform1parms1parms1	env wsse wsu

A sample input document is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
      secext-1.0.xsd">
      <BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
        soap-message-security-1.0#Base64Binary" ValueType="http://docs.oasis-
        open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" wsu:Id="tokenid"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
        1.0.xsd" xmlns="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
        1.0.xsd">MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1
        aVdheTETMBEGA1UEAxMKRXhhbXBsZSBDQTAgFw0xMTExMjExNTQ0MzNaGA8yMTExMTAyODE1NDQz
        M1owNTELMAkGA1UEBhMCVVMxDTALBgNVBAoTBGlXYXkxFzAVBgNVBAMTDkV4YW1wbGUgU2lnbmVy
        MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjWfYtuQS2N6JiznQQappgOPt9Bh60VgDMglei
        xv1Nwz6Tf5CilvwQh1CbbKcQH0q76kmpfys0EurUnZImtNydY8fNwUxJYfyaQxaRtADTQebJK9/8
        2OEtyBSCD9t1QC9YueXcloh2LWi2anGxo2srE1IpsMC5VajbWCvaC+FXQwIDAQABo4GbMIGYMFkG
        A1UdIwRSMFCAFGtf5uFvrr71fnVSluMC7NepZcvAoTWkMzAxMQswCQYDVQQGEwJVUzENMAsGA1UE
        ChMEaVdheTETMBEGA1UEAxMKRXhhbXBsZSBDQYIBDzAdBgNVHQ4EFgQUBzfE1Y2LrHcsMW3zHZ6D
        XnXMzhYwDAYDVR0TAQH/BAIwADAOBgNVHQ8BAf8EBAMCB4AwDQYJKoZIhvcNAQEFBQADgYEAErBW
        S1+hN0BLRVyWdD7orF7sIb3vtCXfzIDbRz6NK6VDoc0k9s7YGpbTlGFcU6Cj1uzCL4Xl7bm3d4vy
        U3D7FWsHvua19/OOIOqMjdxMW2AbNbmsmwAkoU4vi2kQytphSzY2+XCZB174WdB5fZ8VkIiSDmvq
        Imzq7hxGgL3+DuA=</BinarySecurityToken>
    </wsse:Security>
  </env:Header>
  <env:Body wsu:Id="bodyid" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd">...</env:Body>
</env:Envelope>
```

A sample output of the XMLDSigCreateAgent is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
  <env:Header>
    <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
      1.0.xsd">
      <BinarySecurityToken EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-
        message-security-1.0#Base64Binary" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
        200401-wss-x509-token-profile-1.0#X509v3" wsu:Id="tokenid" xmlns:wsu="http://docs.oasis-
        open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns="http://docs.oasis-
        open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
        1.0.xsd">MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1
        aVdheTETMBEGA1UEAxMKRXhhbXBsZSBDQTAgFw0xMTExMjExNTQ0MzNaGA8yMTExMTAyODE1NDQz
        M1owNTELMAkGA1UEBhMCVVMxDTALBgNVBAoTBGlXYXkxFzAVBgNVBAMTDkV4YW1wbGUgU2lnbmVy
        MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCjWfYtuQS2N6JiznQQappgOPt9Bh60VgDMglei
        xv1Nwz6Tf5CilvwQh1CbbKcQH0q76kmpfys0EurUnZImtNydY8fNwUxJYfyaQxaRtADTQebJK9/8
        2OEtyBSCD9t1QC9YueXcloh2LWi2anGxo2srE1IpsMC5VajbWCvaC+FXQwIDAQABo4GbMIGYMFkG
        A1UdIwRSMFCAFGtf5uFvrr71fnVSluMC7NepZcvAoTWkMzAxMQswCQYDVQQGEwJVUzENMAsGA1UE
        ChMEaVdheTETMBEGA1UEAxMKRXhhbXBsZSBDQYIBDzAdBgNVHQ4EFgQUBzfE1Y2LrHcsMW3zHZ6D
        XnXMzhYwDAYDVR0TAQH/BAIwADAOBgNVHQ8BAf8EBAMCB4AwDQYJKoZIhvcNAQEFBQADgYEAErBW
        S1+hN0BLRVyWdD7orF7sIb3vtCXfzIDbRz6NK6VDoc0k9s7YGpbTlGFcU6Cj1uzCL4Xl7bm3d4vy
        U3D7FWsHvua19/OOIOqMjdxMW2AbNbmsmwAkoU4vi2kQytphSzY2+XCZB174WdB5fZ8VkIiSDmvq
        Imzq7hxGgL3+DuA=</BinarySecurityToken>
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
          <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          <ds:Reference URI="#bodyid">
            <ds:Transforms>
              <ds:Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <ds:DigestValue>uoCjbXYPSRb4PAGQ0wZpmQ2PAFY=</ds:DigestValue>
          </ds:Reference>
          <ds:Reference URI="#keyinfoid">
            <ds:Transforms>
              <ds:Transform Algorithm="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
                soap-message-security-1.0#STR-Transform">
                <wsse:TransformationParameters xmlns:wsse="http://docs.oasis-
                  open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
                  <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
                    c14n#">
                    <ec:InclusiveNamespaces PrefixList="env wsse wsu"
                      xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" />
                  </ds:CanonicalizationMethod>
                </wsse:TransformationParameters>
              </ds:Transform>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
            <ds:DigestValue>E/8GktmhFjYFGwBnrXshLWEHiQk=</ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>

          <ds:SignatureValue>fnY2Tn8r4m/EDBzcALDIxoj/COLz4rs6k5a7g3ZO4N27RlpwaaLuRTlUd/gaavk8yGbmJC
          tF6JeE+WnLU0uxLMiOFcXMf7nRkPfL+zyUaat5N72QOTW8kaWaEGe24Ukfot/YAuYGVaJpp+ppok
          pTgkMqarzjCx3xovAiU=</ds:SignatureValue>
        <ds:KeyInfo Id="keyinfoid">
          <wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
            200401-wss-wssecurity-secext-1.0.xsd">
            <wsse:Reference URI="#tokenid" ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
              200401-wss-x509-token-profile-1.0#X509v3" />
          </wsse:SecurityTokenReference>
        </ds:KeyInfo>
      </ds:Signature>
    </wsse:Security>
  </env:Header>
  <env:Body wsu:Id="bodyid" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
    utility-1.0.xsd">...</env:Body>
</env:Envelope>
```

Example 5: Signed Attachment

The following example shows how to sign a MIME attachment. It assumes the Body of the XML document refers to the attachment, and therefore both must be signed. This requires at least two references, one for the Body and another for the attachment. The Reference URI to point to an attachment has the form cid:contentid. The cid URI scheme instructs the Reference to find the attachment with the Content-ID equal to <contentid>. The angle brackets are absent in the URI but must be present in the Content ID header. The Reference must choose one of the attachment transforms. The Attachment Complete Signature Transform (http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Complete-Signature-Transform) indicates that the content and selected headers of the MIME part are referenced for signing, whereas, the Attachment Content Signature Transform (*http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform*) indicates that only the content of a MIME part without headers is referenced for signing.

An attachment can be associated with a document directly from external input (for example, by the NHTTP listener reading an HTTP request), or with the help of one the attachment services (XDAddAttachmentAgent or XDAddAttachmentFromFileAgent). In this example, the XDAddAttachmentFromFileAgent is called to attach a JPEG image to the document. The binary Content-Transfer-Encoding is chosen to take advantage of the more compact format compared to what is possible within an XML document. Notice the explicit angle brackets in the value of the Content-ID parameter. These are part of the Content-ID syntax.

The following table lists the parameter values for the XDAddAttachmentFromFileAgent. Other parameters that are not listed have their default value.

Parameter	Value
Input File	pic.jpg
Content-Type	image/jpeg
Content-Transfer-Encoding	binary
Content-Disposition	attachment; filename=pic.jpg
Content-ID	<pic>

The following table lists the parameter values for the XMLDSigCreateAgent. Other parameters that are not listed have their default value.

Parameter	Value
XML Digital Signature JCE Provider	XMLDSig
Canonicalization Method	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Signature Method	http://www.w3.org/2000/09/xmldsig#rsa-sha1
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
XML Namespace Provider	xmlnsprov
Create Parent Element	true
Signature Parent Element	/soapenv:Envelope/*[1]/self::soapenv:Header/wsse:Security
Include Subject Name	true
Include Certificate Chain	No Certificates
Reference 1 URI	#bodyid
Reference 1 Digest Method	http://www.w3.org/2000/09/xmldsig#sha1
Reference 1 Transform 1	http://www.w3.org/2001/10/xml-exc-c14n#
Reference 2 URI	cid:pic
Reference 2 Digest Method	http://www.w3.org/2000/09/xmldsig#sha1
Reference 2 Transform 1	http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Complete-Signature-Transform

The input document of the XDAddAttachmentFromFileAgent is shown as follows (indented for display purposes only):

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Body xml:id="bodyid">...</env:Body>
</env:Envelope>
```

The output of the XDAddAttachmentFromFileAgent is used as the input of the XMLDSigCreateAgent. When the resulting document is sent by the XDXNHTTPEmitAgent, the HTTP request has the following structure (indented for display purposes only):

```
POST / HTTP/1.1
Content-Length: 95416
Content-Type: multipart/mixed;
   boundary="----=_Part_3_13292599.1303306889045"
Host: localhost:30002
User-Agent: Apache-HttpClient/4.1.1 (java 1.5)

------=_Part_3_13292599.1303306889045
Content-Type: text/xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
            <SignedInfo>
               <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
               <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
               <Reference URI="#bodyid">
                  <Transforms>
                     <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  </Transforms>
                  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <DigestValue>pDa2fftZnR9+lUnjJf0oYtPPLVw=</DigestValue>
               </Reference>
               <Reference URI="cid:pic">
                  <Transforms>
                     <Transform Algorithm="http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Complete-Signature-Transform"/>
                  </Transforms>
                  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <DigestValue>D8DiYaKwL9ngaZvhqzVOsO7T82g=</DigestValue>
               </Reference>
            </SignedInfo>
            <SignatureValue>8Z4Wh00kKjETrBhNdlWY15VjHsz77u0Cu9M0kb1SBk92TGGjqfHAG3KnOoLbG/wlkkOyz+OJiy3C
n4XyS0LLa/SSMARCNlfgeUg1y4GW0LodHUkNJ0bO8eUq67rXP/DU+T+y9lfztWUKHqE93Y5VEKxG
kCGvU0tvHnxrVbi/ma4=</SignatureValue>
            <KeyInfo>
               <X509Data>
                  <X509SubjectName>CN=Example Signer,O=Example,C=US</X509SubjectName>
               </X509Data>
            </KeyInfo>
         </Signature>
      </wsse:Security>
   </env:Header>
   <env:Body xml:id="bodyid">...</env:Body>
</env:Envelope>
------=_Part_3_13292599.1303306889045
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-Disposition: attachment; filename=pic.jpg
Content-ID: <pic>
Content-Length: 93671

<binary image here>
------=_Part_3_13292599.1303306889045--
```

Example 6: Signature Transform Parameters

The following example shows how to specify a complex Signature Transform. In particular, it shows how user parameters are required to specify extra transform parameters. It also proves the transform worked by enabling the debugging aid to view the data digested by each Reference.

The Signature XPath Filter 2.0 Transform takes a sequence of one or more elements named XPath. The text value of the XPath element is an XPath expression. The XPath element has an attribute called Filter that specifies the set operation to apply to the node-set returned by the XPath expression. The possible values are intersect, subtract, or union.

The following is a sample XPath Filter 2.0 transform:

```
<dsig:Signature
    xmlns:dsig="http://www.w3.org/2000/09/xmldsig#"
    xmlns:dsig-xpath="http://www.w3.org/2002/06/xmldsig-filter2">
     <dsig:SignedInfo>
       ...
      <dsig:Reference URI="">
       <dsig:Transforms>
        <dsig:Transform
         Algorithm="http://www.w3.org/2002/06/xmldsig-filter2">
          <dsig-xpath:XPath Filter="intersect">//ToBeSigned</dsig-xpath:XPath>
          <dsig-xpath:XPath Filter="subtract">//NotToBeSigned</dsig-xpath:XPath>
          <dsig-xpath:XPath Filter="union" xmlns:ns1="http://ns1.com">//ns1:ReallyToBeSigned</dsig-xpath:XPath>
        </dsig:Transform>
       </dsig:Transforms>
       ...
      </dsig:Reference>
     </dsig:SignedInfo>
     ...
    </dsig:Signature>
```

In the service configuration, each transform parameter becomes one XPath element. The parameter value consists of the chosen set operation followed by the XPath expression separated by a space. For example, the first XPath element above is configured with the following value:

```
intersect //ToBeSigned
```

If the XPath expression contains XML namespaces, the XML Namespace provider is specified in another related parameter. Because this transform can take more than one parameter, the extra parameters must be entered as user parameters. The name of the parameter contains an index to specify which reference and which transform it belongs to.

To simplify, this example signs the whole document and therefore requires the Enveloped Signature transform. The XPath Filter 2.0 transform becomes the second transform of the first Reference. The location of the Signature element is not specified, therefore it will appear as the last child of the root element.

The following table lists the parameter values of the XMLDSigCreateAgent. Other parameters that are not listed have their default value.

Parameter	Value
XML Digital Signature JCE Provider	XMLDSig
Canonicalization Method	http://www.w3.org/TR/2001/REC-xml-c14n-20010315
Signature Method	http://www.w3.org/2000/09/xmldsig#rsa-sha1
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
Include Subject Name	true
Include Certificate Chain	No Certificates
Reference 1 URI	
Reference 1 Digest Method	http://www.w3.org/2000/09/xmldsig#sha1
Reference 1 Transform 1	http://www.w3.org/2000/09/xmldsig#enveloped-signature
Reference 1 Transform 2	http://www.w3.org/2002/06/xmldsig-filter2
Reference 1 Transform 2 Parameters	intersect //ToBeSigned

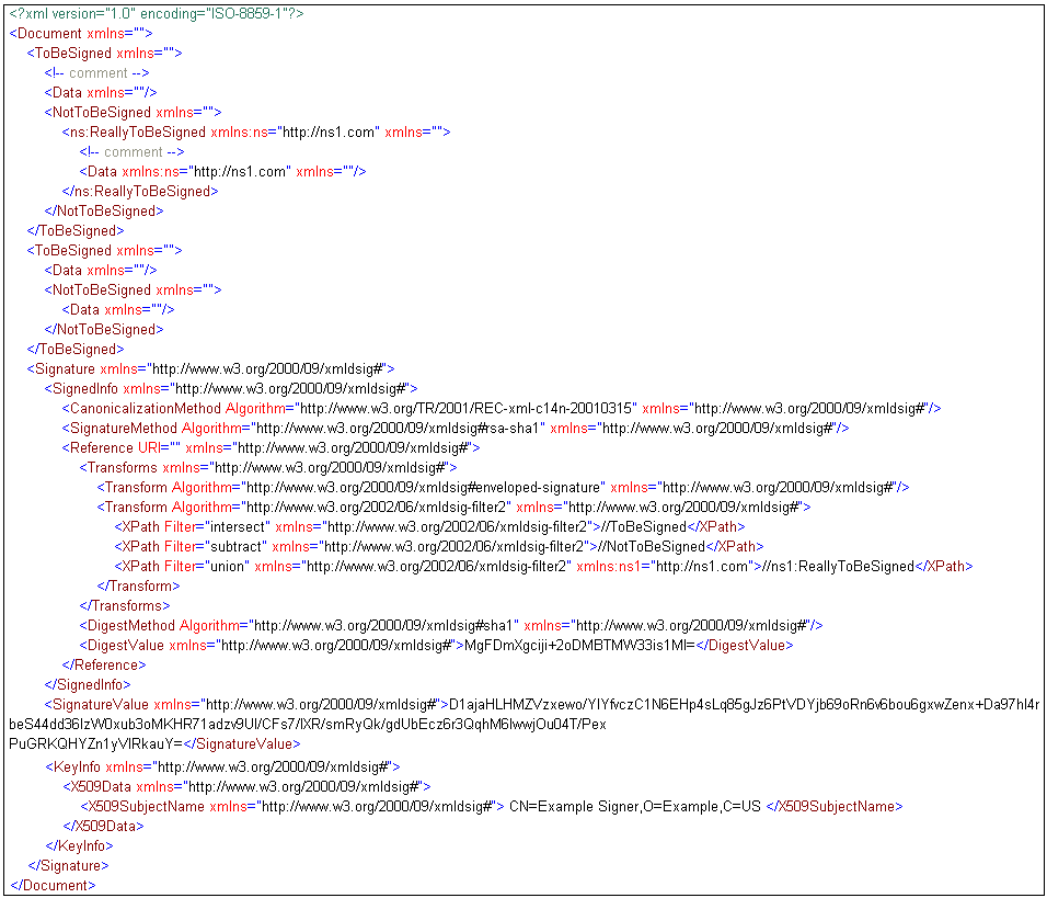
The extra transform parameters are specified with these user parameters. The parameter name and the value must be entered as shown in the following table.

Parameter	Value
ref1transform2parms2	subtract //NotToBeSigned
ref1transform2parms3	union //ns1:ReallyToBeSigned
ref1transform2parms3nsmap	ns1prov

This example assumes there is an XML Namespace provider called ns1prov that defines a single mapping: alias ns1 to namespace http://ns1.com. The input document is shown in the following image (indented for display purposes only):

```
<Document>
  <ToBeSigned>
    <!-- comment -->
    <Data/>
    <NotToBeSigned>
      <ns:ReallyToBeSigned xmlns:ns="http://ns1.com">
        <!-- comment -->
        <Data/>
      </ns:ReallyToBeSigned>
    </NotToBeSigned>
  </ToBeSigned>
  <ToBeSigned>
    <Data/>
    <NotToBeSigned>
      <Data/>
    </NotToBeSigned>
  </ToBeSigned>
</Document>
```

The following image shows the output of the service (indented for display purposes only):



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Document xmlns="">
  <ToBeSigned xmlns="">
    <!-- comment -->
    <Data xmlns=""/>
    <NotToBeSigned xmlns="">
      <ns:ReallyToBeSigned xmlns:ns="http://ns1.com" xmlns="">
        <!-- comment -->
        <Data xmlns:ns="http://ns1.com" xmlns=""/>
      </ns:ReallyToBeSigned>
    </NotToBeSigned>
  </ToBeSigned>
  <ToBeSigned xmlns="">
    <Data xmlns=""/>
    <NotToBeSigned xmlns="">
      <Data xmlns=""/>
    </NotToBeSigned>
  </ToBeSigned>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" xmlns="http://www.w3.org/2000/09/xmldsig#"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" xmlns="http://www.w3.org/2000/09/xmldsig#"/>
      <Reference URI="" xmlns="http://www.w3.org/2000/09/xmldsig#">
        <Transforms xmlns="http://www.w3.org/2000/09/xmldsig#">
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature" xmlns="http://www.w3.org/2000/09/xmldsig#"/>
          <Transform Algorithm="http://www.w3.org/2002/06/xmldsig-filter2" xmlns="http://www.w3.org/2000/09/xmldsig#">
            <XPath Filter="intersect" xmlns="http://www.w3.org/2002/06/xmldsig-filter2">//ToBeSigned</XPath>
            <XPath Filter="subtract" xmlns="http://www.w3.org/2002/06/xmldsig-filter2">//NotToBeSigned</XPath>
            <XPath Filter="union" xmlns="http://www.w3.org/2002/06/xmldsig-filter2" xmlns:ns1="http://ns1.com">//ns1:ReallyToBeSigned</XPath>
          </Transform>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" xmlns="http://www.w3.org/2000/09/xmldsig#"/>
        <DigestValue xmlns="http://www.w3.org/2000/09/xmldsig#">MgFDmXgciji+2oDMBTMW33is1MI=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue xmlns="http://www.w3.org/2000/09/xmldsig#">D1ajaHLHMZVzxewo/YIYfvczC1N6EHp4sLq85gJz6PtVDYjb69oRn6v6bou6gxwZenx+Da97hl4r
beS44dd36IzW0xub3oMKHR71adzv9UI/CFs7/IXR/smRyQk/gdUbEcz6r3QqhM6lwwjOu04T/Pex
PuGRKQHYZn1yVIRkauY=</SignatureValue>
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
      <X509Data xmlns="http://www.w3.org/2000/09/xmldsig#">
        <X509SubjectName xmlns="http://www.w3.org/2000/09/xmldsig#"> CN=Example Signer,O=Example,C=US </X509SubjectName>
      </X509Data>
    </KeyInfo>
  </Signature>
</Document>
```

To prove that the transform worked, you can enable logging at the Tree level. You should see a message similar to the following in your log:

```
TREE (W.file3.1) Digested input for signed reference '':
<ToBeSigned> <Data></Data><ns:ReallyToBeSigned
xmlns:ns="http://ns1.com"><Data></Data></ns:ReallyToBeSigned>
</ToBeSigned><ToBeSigned><Data></Data></ToBeSigned>
```

The Reference URI is shown within single quotes. In this case, the Reference URI was empty because the whole document is signed.

XML Digital Signature Verify Service

Syntax:

`com.ibi.agents.XDXMLDSigVerifyAgent`

Description:

This service is used to validate an XML Digital Signature.

Parameters:

The following tables describe the parameters for the XML Digital Signature Verify service. Each table is followed by a discussion of that parameter group.

Signature Location	
XML Namespace Provider	Provider for the mapping between XML namespace prefix and namespace URI. If left blank, the XPath expression in the Element Path and Required Signature Coverage parameters cannot contain namespaces.
XPath Syntax	Determines which syntax level of XPath should be used. You can select the iWay abbreviated syntax or the XPath 1.0 full syntax. The default option selects the syntax level as set in the General Settings area of the iSM Administration Console.
Signature Element Path	Path to the signature XML element. If left blank, the service will search throughout the document for an element named Signature in the namespace http://www.w3.org/2000/09/xmldsig#.
ID Attributes	Space-separated list of attributes that are considered type ID. The value of an ID attribute can be used in a same document reference with a URI of the form #idvalue. Each attribute declaration has the form ns:*/@ns1:attrib or @ns1:attrib where ns: and ns1: are optional. If used, the ns and ns1 prefixes must be declared in the XML Namespace Provider. The form @ns1:attrib means an Attribute named attrib in XML Namespace ns1. The form ns:*/@ns1:attrib is similar except the attribute must also appear on an element of any name in the XML Namespace ns. The default value is: `xml:id ds:*/@Id wsu:Id xenc:Id`

Signature Location	
Remove Security Parent Element	If set to *true*, the WSSE Security parent element is removed from the document after the verification is successful.

The Signature Element Path parameter is an XPath expression that evaluates to the Signature node. For example, for a SOAP message, the XPath expression would be a path somewhere within the SOAP Headers. The XPath expression can be the union of two paths if the application is expecting signatures in one of two locations. An XML Namespace Provider is needed if the XPath expression contains namespaces.

The Remove Security Parent Element boolean parameter is a way to remove a SOAP header from the document after the verification is successful.

Acceptance Criteria	
XML Digital Signature JCE Provider	JCE Provider for the XMLSignatureFactory service.
TrustStore Provider	Provider for the keystore containing the Certificate Authorities.
Certificate Store Providers	Comma-separated List of Keystore, Directory CertStore or LDAP providers for the certificate stores used to complete signer certificate chains when the signature contains fewer certificates than needed.
PKIX Signature JCE Cryptography Provider	JCE Provider for Signature Objects created by the PKIX Certificate Path Builder.
PKIX JCE Provider	JCE Provider for PKIX services. If left blank, the default JCE provider for PKIX will be used.
Enable Certificate Revocation	If set to true, use the CRLs from the CertStore to check whether the certificate of the signer has been revoked.
Enforce KeyUsage Extension	If set to true, verify certificates used for signing allow the digitalSignature KeyUsage extension. Select *true* or *false* (default) from the drop-down list.

Acceptance Criteria	
Acceptable Transforms	Space-separated list of transforms that can appear in the XML Digital Signature. Other transforms will cause a validation failure before being evaluated. If this field is left blank, all transforms are accepted.
Required Signature Coverage	An XPath expression that returns a node-set, where each node in the set must have been signed by the Signature to be considered valid.
Unsigned Attachment	Action to perform when a document contains an unsigned attachment. Select one of the following values from the drop-down list: ❑ Keep Unsigned Attachment {keep} (default) ❑ Remove Unsigned Attachment {remove} ❑ Fail Validation {fail}

The XML Digital Signature JCE Provider selects the XML Digital Signature implementation. Choose *XMLDSig* for the implementation built-in to Java, or *ApacheXMLDSig* for the Apache Santuario implementation. ApacheXMLDSig is often a better choice because it is updated more frequently and therefore is more up to date.

The service recovers the signer public key based on the information it finds in the KeyInfo element. To begin, the service collects all the X509Certificates and X509CRLs under the X509Data element and creates a certificate store. This store together with the certificate store providers will be used to complete the certificate chain. The service then iterates through the KeyInfo content. The service understands X509IssuerSerial, X509SubjectName and X509SKI (for example, the Subject Key Identifier). The service also understands a wsse:SecurityTokenReference pointing to a wsse:BinarySecurityToken holding an X509 certificate encoded in base64. The service iterates in order of appearance and works with the first item it understands ignoring subsequent ones. A Certificate selector is created and the CertStores are queried to complete and validate the chain.

Edges:

The following table lists and describes the edges that are returned by the XML Digital Signature Verify service.

Edge	Description
success	The Signature is valid.
fail_parse	An iFL or XPath expression could not be evaluated.
fail_operation	The validation could not be performed.
fail_unsigned	The XML document is not signed.
fail_verify	The Signature is invalid.
fail_coverage	The application specified a node or attachment that should have been signed but was not covered by the Signature.

If the signature validates, the service will continue on the success edge. If validation fails because of Unsigned Attachments or incomplete Required Signature Coverage, the service will follow the fail_coverage edge. If validation fails for other reasons, the service will follow the fail_verify edge. If there is no signature, the flow will continue on the fail_unsigned edge.

When the Tree level is selected in the trace settings, the service will log the referenced data that was actually digested. It will also show whether core validation passed, and the validation status of each reference.

Special Registers

The following table lists and describes the special registers assigned upon successful validation of the signature.

Special Register	Description
xmldsig_signer	Holds the signer Distinguished Name.
xmldsig_signer_cn	Holds the signer Common Name found in the signer Distinguished Name.

Examples

The examples in this section are specific to the XML Digital Signature Verify service (com.ibi.agents.XDXMLDSigVerifyAgent). For your convenience, the sample input and output documents are attached to the PDF in unabbreviated and unindented form.

For PDF-compatibility purposes, the file extension of the *XMLDSigVerify.zip* file is temporarily renamed to *.zap*. After saving this file to your file system, you must rename this extension back to *.zip* before the file can be used.

Example 1: Completing the Certificate Chain

The verification of a digital signature involves the following steps:

1. The Signature element is retrieved and its syntax is checked.
2. For each Reference, its transforms are evaluated and the output digested into a hash. The hash is compared against that Reference DigestValue in the signature.
3. The signer certificate is retrieved and the certificate chain is validated.
4. The signature is recomputed over the SignedInfo and compared against the SignatureValue in the document.

These steps can be performed by the XDXMLDSigVerifyAgent with very little configuration because the signature contains the necessary information. This makes the service more adaptable to different signature formats it may receive. By default, the service searches the whole document for an element named Signature in the namespace *http://www.w3.org/2000/09/xmldsig#*.

The Reference URI instructs the service where to find the referenced data. The signature declares the algorithms used for hashing and signing. The KeyInfo element provides the hints to find the signer certificate. The only necessary configuration is the TrustStore used to validate the certificate chain. The TrustStore parameter gives the name of the KeyStore provider that declares a KeyStore file containing the certificates of the Trusted CAs. It is highly recommended to use this KeyStore exclusively as a TrustStore. It should not contain private keys or any other unrelated certificates. The JRE includes a sample TrustStore, which is located in the following directory:

```
<java-home>/lib/security/cacerts
```

This file is convenient to retrieve the certificates of some well known CAs. It is not recommended to use the JRE sample TrustStore directly. It is better to restrict the contents of the TrustStore to the small list of Trusted CAs the application is willing to trust.

In this example, the flexibility of the service is demonstrated by validating various signatures showing multiple ways the certificate chain can be completed.

The following table lists the parameter values for the XDXMLDSigVerifyAgent. Other parameters that are not listed have their default value.

Parameter	Value
TrustStore Provider	trustprov

This assumes trustprov is the name of a KeyStore provider that contains the certificates of the Trusted CAs.

The simplest case is when the Signature contains the complete certificate chain, though this produces documents that are slightly bulkier. The certificates appear in X509Certificate elements under the KeyInfo in no particular order. The verify service needs an extra hint to determine which one is the signer certificate. For example, the SubjectName can also be added to the KeyInfo.

The XDXMLDSigCreateAgent can create this signature by setting the Include Certificate Chain parameter to *Complete Certificate Chain* and setting the Include Subject Name parameter to *true*.

In this sample signature, the certificate chain contains two certificates: the signer certificate and the issuer CA. The TrustStore must contain the certificate of the issuer CA to validate the chain.

A sample input document is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Document>
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
         <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <Reference URI="">
            <Transforms>
               <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>3xGw/Hx0YFiyrRjWBdb9P7Pr0A=</DigestValue>
         </Reference>
      </SignedInfo>
      <SignatureValue>
cDI2ZRHHLWvrlVGYYAI1l/liCYmqHFwwSnjKMlHtJxzhkFf7rGJSbNCctorfS8lWLNhnahBn37NU
8TC4/pXYe1qv3TKjt+B8fc1UDR/tLDrNdo6A5ihwvAq3zZAl7/m1q9tRfK0X0Q6EPexH1gtV0vlw
4CTBcJgznyv2Oo24OrM=</SignatureValue>
      <KeyInfo>
         <X509Data>
            <X509SubjectName>CN=Example Signer,O=Example,C=US</X509SubjectName>
            <X509Certificate>
MIICmDCCAgGgAwIBAgIBGDANBgkqhkiG9w0BAQUFADA6MQswCQYDVQQGEwJVUzEQMA4GA1U
EChMH
...</X509Certificate>
            <X509Certificate>
MIIB5TCCAU4CARQwDQYJKoZIhvcNAQEFBQAwOjELMAkGA1UEBhMCVVMxEDAOBgNVBAoTB1B
hcnRu
...</X509Certificate>
         </X509Data>
      </KeyInfo>
   </Signature>
   <Body>...</Body>
</Document>
```

Since the issuer CA is already in the TrustStore, many applications choose to omit the CA certificate in the signature KeyInfo. The service is able to complete the certificate chain because it adds the TrustStore to the list of CertStores to search for certificates. The service still needs an extra hint to recognize that the certificate in the signature is the signer certificate. This example includes the certificate IssuerName/SerialNumber in the KeyInfo. In general, the IssuerName/SerialNumber is better than the Subject DN because it uniquely identifies the certificate (even if it is renewed later).

The XDXMLDSigCreateAgent can create this signature by setting the Include Certificate Chain parameter to *Signer Certificate Only*, the Include Issuer Serial parameter to *true*, and the Include Subject Name parameter to *false*.

A sample input document is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Document>
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
         <CanonicalizationMethod Algorithm="
http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <Reference URI="">
            <Transforms>
               <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>3xGw//Hx0YFiyrRjWBdb9P7Pr0A=</DigestValue>
         </Reference>
      </SignedInfo>
      <SignatureValue>
cDI2ZRHHLWvrlVGYYAI1l/liCYmqHFwwSnjKMlHtJxzhkFf7rGJSbNCctorfS8lWLNhnahBn37NU
8TC4/pXYe1qv3TKjt+B8fc1UDR/tLDrNdo6A5ihwwAq3zZAl7/m1q9tRfK0X0Q6EPexH1gtV0vlw
4CTBcJgznyv2Oo24OrM=</SignatureValue>
      <KeyInfo>
         <X509Data>
            <X509IssuerSerial>
               <X509IssuerName>CN=Example
CA,O=Example,C=US</X509IssuerName>
               <X509SerialNumber>24</X509SerialNumber>
            </X509IssuerSerial>
            <X509Certificate>
MIICmDCCAgGgAwIBAgIBGDANBgkqhkiG9w0BAQUFADA6MQswCQYDVQQGEwJVUzEQMA4
GA1UEChMH
...</X509Certificate>
         </X509Data>
      </KeyInfo>
   </Signature>
   <Body>...</Body>
</Document>
```

Some web service security standards advocate that the certificate should be passed in a BinarySecurityToken within the SOAP Headers. The KeyInfo element should contain a SecurityTokenReference pointing to the BinarySecurityToken. The service is able to follow the URI in the SecurityTokenReference to retrieve the certificate in the BinarySecurityToken. It is recommended to include the BinarySecurityToken within the signed data by declaring a signature Reference that covers that element.

For more information on how to create this signature, see *Example 3: WSSE SecurityTokenReference* on page 115.

A sample input document is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
wss-wssecurity-secext-1.0.xsd">
         <BinarySecurityTokenEncoding Type="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsssoapmessage-
security-1.0#Base64Binary" ValueType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" wsu:Id="tokenid"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-secext-1.0.xsd">
MIICmDCCAgGgAwIBAgIBGDANBgkqh
kiG9w0BAQUFADA6MQswCQYDVQQGEwJVUzEQMA4GA1UEChMH</BinarySecurityTokenEncoding>
         <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
            <SignedInfo>
               <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
               <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
               <Reference URI="#tokenid">
                  <Transforms>
                     <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  </Transforms>
                  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <DigestValue>duT/tcXl+1fbuxyXZHeEBrMy+Fs=</DigestValue>
               </Reference>
               <Reference URI="#bodyid">
                  <Transforms>
                     <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  </Transforms>
                  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <DigestValue>XLnuUJo+oiZ/KnAfleCuMqNfRfM=</DigestValue>
               </Reference>
            </SignedInfo>
            <SignatureValue>
gpUiN0PA/YEIYhXVaQcdkacKPCdBBB28nH3Yf6lpMnFmwJkkYAVXAhPLEkR48o
tTAtfnOIK2go0VwhnniD0SXUg53WNWfhxp4xUWvZASreygVLXLs83l5NnsR0WQKZxRR/9T3Qo7DzsOetDWm
HyTehj
bVMJMKwWRSGHk+N9DI=</SignatureValue>
            <KeyInfo>
               <wsse:SecurityTokenReference xmlns:wsse="http://docs.oasis
-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
                  <wsse:Reference URI="#tokenid" ValueType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
               </wsse:SecurityTokenReference>
            </KeyInfo>
         </Signature>
      </wsse:Security>
   </env:Header>
   <env:Body wsu:Id="bodyid" xmlns:wsu="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"/>
</env:Envelope>
```

Example 2: Omitting the Certificate Chain

If the originator is sending many signed documents to the application, it might be more efficient to send the certificate chain out of band and let the application retrieve it locally. The signer certificate must be added to one of the CertStores accessible to the service. Intermediate CAs must also be added to the CertStores if the chain has a depth of 3 or more. The Trusted CA must be in the TrustStore as always. The service uses the hints in the KeyInfo to query its CertStores to find the signer certificate. The service completes the chain by querying for the parent certificates.

The Certificate Store Providers parameter accepts a comma-delimited list of Keystore, Directory CertStore, or LDAP providers. The TrustStore is implicitly used as a CertStore so the Trusted CA certificates are not required to be duplicated in one of those CertStores. A certificate can be added to a Keystore using the JDK keytool. To add a certificate to a Directory CertStore, save the certificate file in the file system directory. To add a certificate to LDAP, consult your system administrator. The LDAP database must conform to the schema described in RFC2587.

The following table lists the parameter values for the XDXMLDSigVerifyAgent. Other parameters that are not listed have their default value.

Parameter	Value
TrustStore Provider	trustprov
Certificate Store Providers	ksprov

This assumes that trustprov is the name of a KeyStore provider that contains the certificates of the Trusted CAs and that ksprov is the name of a KeyStore provider that points to a keystore file that contains the signer certificate. The Certificate Store provider could also be a Directory CertStore or an LDAP Provider.

The XDXMLDSigCreateAgent can create this signature by setting the Include Certificate Chain parameter to *No Certificates*, the Include Issuer Serial parameter to *true*, and the Include Subject Name parameter to *false*.

A sample input document is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Document>
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
         <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
         <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <Reference URI="">
            <Transforms>
               <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
            <DigestValue>3xGv//Hx0YFiyrRjWBdb9P7Pr0A=</DigestValue>
         </Reference>
      </SignedInfo>
      <SignatureValue>
cDI2ZRHHLWvrlVGYYAI1l/liCYmqHFwwSnjKMlHtJxzhkFf7rGJSbNCctorfS8lWLNhnahBn37NU
8TC4/pXYe1qv3TKjt+B8fc1UDR/tLDrNdo6A5ihwvAq3zZAI7/m1q9tRfK0X0Q6EPexH1gtV0vlw
4CTBcJgznyv2Oo24OrM=</SignatureValue>
      <KeyInfo>
         <X509Data>
            <X509IssuerSerial>
               <X509IssuerName>CN=Example Signer,O=Example,C=US</X509IssuerName>
               <X509SerialNumber>24</X509SerialNumber>
            </X509IssuerSerial>
         </X509Data>
      </KeyInfo>
   </Signature>
   <Body>...</Body>
</Document>
```

Example 3: Certificate Revocation

For a variety of reasons, the certificate issuer might revoke a certificate before its expiration date. For example, this can happen if the private key is compromised, an employee has left the company, and so on. The issuer publishes a Certificate Revocation List (CRL) to instruct applications that these certificates are no longer valid. The service can validate the signer certificate chain against a CRL if Certificate Revocation is enabled. The service will query its CertStores for the CRL from that issuer. If there are no revoked certificates, a CRL that contains no certificates is required. If the CRL is not found, then the service takes the conservative approach and assumes that the certificate might be invalid and rejects the signature as a result. Since a KeyStore cannot contain a CRL, the CRL must be stored in another provider, such as a Directory CertStore or LDAP. The simplest solution is to store the CRL file in the file system directory of a Directory CertStore.

The KeyInfo element may also contain X509CRL elements to include CRLs within the Signature. The service can use the CRLs in X509CRL elements if present. In practice, this is rarely used since CRLs can become quite large compared to the size of the message.

The following table lists the parameter values for the XDXMLDSigVerifyAgent. Other parameters that are not listed have their default value.

Parameter	Value
TrustStore Provider	trustprov
Certificate Store Providers	ksprov,certprov
Enable Certificate Revocation	true

This assumes that trustprov is the name of a KeyStore provider that contains the certificates of the Trusted CAs; ksprov is the name of a KeyStore provider that points to a keystore file that contains the signer certificate; and certprov is the name of a Directory CertStore provider that points to a directory that contains the issuer CRL in a file.

Any signature example you have seen so far can be validated by this service. The signature will fail if the signer certificate was revoked, and succeed otherwise.

Example 4: Signature Coverage

A signature can be valid and still be useless. The application must also verify that all sensitive information in the document has been covered by the signature. Otherwise, the information could be modified after the fact without affecting the signature. The service can verify the signature has appropriate coverage.

The Required Signature Coverage parameter is an XPath expression that returns a NodeSet. Each node in the NodeSet must have been signed to consider the signature valid. For every sensitive node, the XPath expression should return that node or one of its ancestors. Notice that the parameter does not demand the presence of a node. If a node is missing in the document, then it does not matter whether that node has been signed or not because there is no possibility the application could use that node anyway. The application can reject the incomplete message at a later time, but that is unrelated to the signature.

For example, if the application expects a SOAP message with a BinarySecurityToken in a SOAP Header, it can specify the union of the path to the security token and the SOAP Body in the XPath expression. Since a Signature never covers itself, the Signature element should never be returned by the XPath expression.

To avoid the dual evaluation of the transforms, that feature assumes the transforms are hierarchical starting at the node pointed to by the Reference URI. The inclusive and exclusive XML canonical transforms are supported, but the XPath Filter and XSLT transforms are not.

If the application accesses the attachments, then it should ensure they have been covered by the signature. The Unsigned Attachment parameter determines what action to perform when there are unsigned attachments. The service can keep the unsigned attachments and accept the signature, remove the unsigned attachments and accept the signature, or fail validation. There is no option to demand the presence of an attachment since there is no possibility the application could use an unsigned attachment if that attachment is absent. Again, the application can reject incomplete messages at a later time.

It is assumed that the XML Namespace provider xmlnsprov defines the following prefixes:

Prefix	XML Namespace
soapenv	*http://schemas.xmlsoap.org/soap/envelope/*
wsse	*http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd*

The following table lists the parameter values for the XDXMLDSigVerifyAgent. Other parameters that are not listed have their default value.

Parameter	Value
TrustStore Provider	trustprov

Parameter	Value
XML Namespace Provider	xmlnsprov
Required Signature Coverage	/soapenv:Envelope/soapenv:Header/ wsse:Security/wsse:BinarySecurityToken| /soapenv:Envelope/soapenv:Body
Unsigned Attachment	Fail Validation

The service will fail the following signature because the BinarySecurityToken in the SOAP Header is not covered by any Reference. The Signature would have been valid if the Required Signature Coverage parameter had been left blank.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header>
      <wsse:Security xmlns:wsse="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
         <BinarySecurityToken EncodingType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0#Base64Binary" ValueType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3" wsu:Id="tokenid"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wsswssecurity-
secext-1.0.xsd">MIICmDCCAgGgAwIBAgIBGDANBgkqhkiG9w0
BAQUFADA6MQswCQYDVQQGEwJVUzEQMA4GA1UEChMH
...</BinarySecurityToken>
         <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
            <SignedInfo>
               <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
               <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
               <Reference URI="#bodyid">
                  <Transforms>
                     <Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                  </Transforms>
                  <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
                  <DigestValue>uoCjbXYPSRb4PAGQ0wZpmQ2PAFY=</DigestValue>
               </Reference>
            </SignedInfo>
            <SignatureValue>
XAuBDKExkpOZgSAXmDZjkLMil/VmaN49rPmtSLw8075GDsFde3DrDz01YxGy1qv3uqIYRCsab1nG
+R4W5sp0YC2Vk++gyrd2etvurp3IfaRFkXvWKSImou+RSgo35jFTHorR8uGyg4IZf0bdw7kxBCOC
9m7EMYarjcKfMsWwvc=</SignatureValue>
            <KeyInfo>
               <wsse:SecurityTokenReference xmlns:wsse="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
                  <wsse:Reference URI="#tokenid" ValueType="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3"/>
               </wsse:SecurityTokenReference>
            </KeyInfo>
         </Signature>
      </wsse:Security>
   </env:Header>
   <env:Body wsu:Id="bodyid" xmlns:wsu="
http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd">...</env:Body>
</env:Envelope>
```

Example 5: Reducing Risks

By default, the service looks throughout the document for the Signature element. A clever attacker could add a second Signature to confuse the service. It is more secure to declare the location where the application expects to find the signature. The Signature Element Path parameter is an XPath expression that evaluates to the Signature node. For example, for a SOAP message, the XPath expression would be a path somewhere within the SOAP Headers. The XPath expression can be the union of two paths if the application is expecting signatures in one of two locations. An XML Namespace Provider is needed if the XPath expression contains namespaces.

Some transforms are very powerful, for example, the XSLT transform describes an engine that is highly programmable. This makes the signature verification process vulnerable because it is the equivalent of running code provided by the peer. An attacker can easily write a transform that adversely affects the server, for example by running an infinite loop. To counter these threats, the Acceptable Transforms parameter can be set to a space-separated list of transform names. The service will accept to execute a transform only if it appears in this white list. If another transform is encountered, then the validation fails before the transform is evaluated. By default, the service accepts to execute any transform available to the server. The service treats the Signature canonicalization method as just another kind of transform with respect to this parameter, and therefore its algorithm must also be present in the white list.

The service can verify an encryption key has not been used inadvertently for signing. When the Enforce KeyUsage Extension parameter is set, the service checks the KeyUsage extension in the signer certificate for the presence of the digitalSignature and/or nonRepudiation flags. The validation fails if the KeyUsage extension is missing or the chosen combination of the flags is off. An application interested in these features will also be interested in the Signature Coverage feature, which is discussed in *Example 4: Signature Coverage* on page 143.

It is assumed that the XML Namespace provider xmlnsprov defines the following prefixes:

Prefix	XML Namespace
soapenv	*http://schemas.xmlsoap.org/soap/envelope/*
wsse	*http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd*
ds	*http://www.w3.org/2000/09/xmldsig#*

The following table lists the parameter values for the service. Other parameters that are not listed have their default value.

Parameter	Value
TrustStore Provider	trustprov
Enforce KeyUsage Extension	digitalSignature
XML Namespace Provider	xmlnsprov
Signature Element Path	/soapenv:Envelope/soapenv:Header/ wsse:Security/ds:Signature
Acceptable Transforms	*http://www.w3.org/TR/2001/REC-xml-c14n-20010315* *http://www.w3.org/2000/09/xmldsig#enveloped-signature* *http://www.w3.org/2001/10/xml-exc-c14n#* *http://docs.oasis-open.org/wss/oasis-wss-SwAProfile-1.1#Attachment-Content-Signature-Transform*

The following signature is rejected by the service because it is not in the expected location within the document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Document>
  <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>3xGw//Hx0YFiyrRjWBdb9P7Pr0A=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>
cDI2ZRHHLWvrlVGYYAI1I/liCYmqHFwwSnjKMlHtJxzhkFf7rGJSbNCctorfS8IWLNhnahBn37NU
8TC4/pXYe1qv3TKjt+B8fc1UDR/tLDrNdo6A5ihwwAq3zZAI7/m1q9tRfK0X0Q6EPexH1gtV0vlw
4CTBcJgznyv2Oo24OrM=</SignatureValue>
    <KeyInfo>
      <X509Data>
        <X509IssuerSerial>
          <X509IssuerName>CN=Example Signer,O=Example,C=US</X509IssuerName>
          <X509SerialNumber>24</X509SerialNumber>
        </X509IssuerSerial>
      </X509Data>
    </KeyInfo>
  </Signature>
  <Body>...</Body>
</Document>
```

The following signature is rejected because the Reference includes a transform that is not listed in the list provided by the Acceptable Transforms parameter:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Document xmlns="">
   <Signature xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
         <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"
xmlns="http://www.w3.org/2000/09/xmldsig#"/>
         <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" xmlns="
http://www.w3.org/2000/09/xmldsig#"/>
         <Reference URI="" xmlns="http://www.w3.org/2000/09/xmldsig#">
            <Transforms xmlns="http://www.w3.org/2000/09/xmldsig#">
               <Transform Algorithm="http://www.w3.org/TR/1999/REC-xpath-19991116" xmlns="
http://www.w3.org/2000/09/xmldsig#">
                  <XPath xmlns:dsig="http://www.w3.org/2000/09/xmldsig#" xmlns="
http://www.w3.org/2000/09/xmldsig#">not(ancestor-or-self::dsig:Signature)</XPath>
               </Transform>
            </Transforms>
            <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" xmlns="
http://www.w3.org/2000/09/xmldsig#"/>
            <DigestValue xmlns="http://www.w3.org/2000/09/xmldsig#">
3xGw//Hx0YFiyrRjWBdb9P7Pr0A=</DigestValue>
         </Reference>
      </SignedInfo>
      <SignatureValue xmlns="http://www.w3.org/2000/09/xmldsig#">
hkdH5hR/JbLsV/ploeccd7rWZzll0v
ZSmivaVrr/UwaxdZJhLep59pDD1ennGQQkJZB0b0/iVzc4
lcCatmOlkguZzpptKyL+QAwdUjvuVfiXHLB3RYqy9wZPaX8UNmtqGHG6jnGtaoMrrb9CmFhzlZMr
hvPZFYgQhZzPUT2YG0Y=</SignatureValue>
      <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
         <X509Data xmlns="http://www.w3.org/2000/09/xmldsig#">
            <X509SubjectName xmlns="http://www.w3.org/2000/09/xmldsig#">CN=Example
Signer,O=Example,C=US</X509SubjectName>
         </X509Data>
      </KeyInfo>
   </Signature>
   <Body xmlns="">...</Body>
</Document>
```

XAdES Digital Signature Create Service

Syntax:

`com.ibi.agents.XDXAdESCreateAgent`

Description:

This service is used to generate an XML Advanced Electronic Signature. XAdES defines formats for XML Digital Signatures that remain valid over long periods and incorporate additional useful information in common use cases. XAdES was developed by the European Telecommunications Standards Institute and published in the Technical Specification ETSI TS 101 903. Individual copies of this specification can be downloaded from *http://www.etsi.org*. This guide assumes that the reader is familiar with the XAdES specification.

An XAdES signature is a regular XML Digital Signature with extra signed and unsigned properties. The specification is organized in a handful of signature forms that define strictly increasing set of properties from the simplest form to the most complex one. The specification uses the term *qualifying properties* because they qualify the signature, the signer, all references, or specific references. A SignedDataObject is an object referenced by an XML Digital Signature Reference.

Parameters:

The following tables describe the parameters for the XAdES Digital Signature Create service. Each table is followed by a discussion of that parameter group.

Algorithms	
XAdES Form	The signature form determines which signed and unsigned properties are added to the signature. The options are XAdES-BES (Basic Electronic Signature), XAdES-EPES (Explicit Policy Electronic Signature), XAdES-T (electronic signature with Time), XAdES-C (electronic signature with Complete validation data references).
Signature Method	Signature algorithm used to convert the canonicalized SignedInfo into the SignatureValue.
Signature Canonicalization Method	Algorithm used to canonicalize the SignedInfo element before it is digested as part of the signature operation.
Reference Digest Method	Digest algorithm applied to the data object references to yield the DigestValue.

Algorithms	
Reference Property Digest Method	Digest algorithm applied to the qualifying properties that contain references to certificates, CRLs, and so on.
Time Stamp Canonicalization Method	Algorithm used to canonicalize the qualifying properties, when needed by a time stamp.
Time Stamp Digest Method	Digest algorithm applied to the qualifying properties that contain time stamps.
Message Digest JCE Provider	JCE Provider for the MessageDigest service.

The signature form determines which signed and unsigned properties are added to the signature. The forms are organized in a hierarchy where each form augments the previous form with more properties.

The Signature Method is the Signature algorithm used to convert the canonicalized SignedInfo into the SignatureValue. Notice the SHA1 algorithm is common, but is not considered secure anymore.

The Signature Canonicalization Method is the Algorithm used to canonicalize the SignedInfo element before it is digested as part of the signature operation.

The Reference Digest Method is the algorithm used to hash the references. The same digest method will be used by all the references.

The Reference Property Digest Method is the algorithm used to hash some values in XAdES properties. For example, the CertDigest of the SigningCertificate, or the SigPolicyHash of the SignaturePolicyIdentifier.

The Time Stamp Canonicalization Method is the algorithm used to canonicalize the data for a time stamp. The Time Stamp Digest Method is the hash algorithm to reduce the data before it is signed by a time stamp. The Message Digest JCE Provider is the JCE Provider used to create the JCE MessageDigest instance.

Signature Key	
KeyStore Provider	Provider for the keystore containing the signature private key.

Signature Key	
Signing Key Alias	Private key alias used to sign the SignedInfo.
Signing Key Password	Password for the signing private key. If left blank, the password for accessing the keystore will be used.

The KeyStore Provider is the name of the provider that holds the private key. The Signing Key Alias and Signing Key Password are the Alias and Password for the private key. This key must be compatible with the signature algorithm chosen in the Signature Method parameter. The service will enforce the digitalSignature or the nonRepudiation usage if the KeyUsage extension is present in the Signing Key Certificate.

Signature Location	
XML Namespace Provider	Provider for the mapping between XML namespace prefix and namespace URI. If left blank, the XPath expression in the Signature Parent Element cannot contain namespaces.
XPath Syntax	Determines which syntax level of XPath should be used. You can select the iWay abbreviated syntax or the XPath 1.0 full syntax. The default option selects the syntax level as set in the General Settings area of the iSM Administration Console.
Create Parent Element	Determines whether the parent element is created if it is missing. Select *true* or *false* (default) from the drop-down list.

Signature Location	
Signature Parent Element	Path to the element where the signature will be inserted. If left blank, then the signature parent is the root element. If the Create Parent Element parameter is set to *true*, then the expression must adhere to Restricted XPath syntax, otherwise the expression may adhere to the full syntax of the XPath engine selected by the XPath Syntax parameter. Restricted XPath has the form /step1/step2/... where a step has the form ns:elem[predicate] or a pair of consecutive steps that has the form *[1]/ self::ns:elem[predicate] to indicate the element must be the first child of its parent. The namespace prefixes are optional, but if present they must be declared in the XML Namespace provider. The predicate is optional, but when present it has the form [@ns1:attr1='val1' and @ns2:attr2='val2' and ...]. If no element matches the Restricted XPath expression and the Create Parent Element parameter is set to true, then the necessary elements and attributes will be created, such that the expression would match successfully.

The XML Namespace Provider is optional. It is the name of the provider that gives the mapping between XML Namespace prefixes and XML Namespace URIs. The Signature Parent Element is an XPath expression pointing to the element where the ds:Signature element will be inserted. This expression cannot contain namespace prefixes if the XML Namespace Provider is left blank.

When the Create Parent Element parameter is true, the parent element will be created if needed, but the XPath expression must adhere to the Restricted XPath syntax. When the Create parent Element parameter is false, the parent element must exist but the expression may adhere to the full syntax of the XPath engine selected by the XPath Syntax parameter.

KeyInfo	
Include Signing Certificate	Indicates whether the signing certificate should be included in a ds:X509Certificate element within ds:KeyInfo.
Include Public Key	Indicates whether a ds:KeyValue element containing the value of the public key should be included in ds:KeyInfo.

These parameters determine the content of the generated KeyInfo element. They can be used in any combination. If none of the parameters are used, the KeyInfo element will not appear. Since the KeyInfo is not signed in general, the digest of every certificate in the certificate chain will also appear under the SigningCertificate property. The SigningCertificate is a signed qualifying property that is always added to the XAdES signature.

The Include Signing Certificate boolean parameter determines whether the signing certificate is included in the KeyInfo element. If so, it will appear base64 encoded in a KeyInfo/X509Data/X509Certificate element.

The Include Public Key boolean parameter determines whether the public key is included in the KeyInfo element. For an RSA key, this adds a KeyInfo/KeyValue/RSAKeyValue/Modulus element encoded in base 64.

Qualifying Properties	
All Signed Data Objects Commitment	Commitment type that applies to all the signed data objects.
All Signed Data Objects Commitment Description	The text description for the commitment type that applies to all the signed data objects. A default English description will be used if a standard commitment type is chosen and this property is left blank.

Qualifying Properties	
All Signed Data Objects Time Stamp	Determines whether to add a time stamp computed before the signature production, over the sequence formed by ALL the Reference elements within the SignedInfo referencing whatever the signer wants to sign except the SignedProperties element.
Sign Signing Certificate	Indicates whether the signature should cover the ds:X509Certificate element containing the signing certificate. This is only considered if Include Signing Certificate is selected.
Signing Time	Specifies the time at which the signer purportedly performed the signing process. Leave blank to use the current time.
Signer Roles	A newline separated list of the roles claimed by the signer.
TSA URL	The location of the Time Stamp Authority used to create time stamps.

These parameters define global qualifying properties of the signature, the signer or all of the references. See the reference parameters for reference specific qualifying properties.

The All Signed Data Objects Commitment identifies the type of commitment made by the signer with respect to all the references. It is possible to use custom commitment types by typing a custom ObjectIdentifier. The dropdown list contains the commitment types already defined by the XAdES Technical Specification, namely:

- **Proof of origin.** Indicates that the signer recognizes to have created, approved, and sent the signed data object.
- **Proof of receipt.** Indicates that signer recognizes to have received the content of the signed data object.
- **Proof of delivery.** Indicates that the TSP providing that indication has delivered a signed data object in a local store accessible to the recipient of the signed data object.
- **Proof of sender.** Indicates that the entity providing that indication has sent the signed data object (but not necessarily created it).

- **Proof of approval.** Indicates that the signer has approved the content of the signed data object.
- **Proof of creation.** Indicates that the signer has created the signed data object (but not necessarily approved, nor sent it).

The All Signed Data Objects Commitment Description parameter contains a human readable description of the commitment type. Enter the text of the custom description. This property can also be left blank when a standard commitment type is chosen, and a default English description will be used. This parameter is ignored if the All Signed Data Objects Commitment parameter is unspecified.

The All Signed Data Objects Time Stamp Boolean parameter indicates whether the AllDataObjectsTimeStamp element is generated. This element contains the time stamp computed before the signature production, over the sequence formed by processing all the References except the Reference to the SignedProperties.

The Sign Signing Certificate Boolean parameter indicates whether the signature should cover the X509Certificate element containing the signing certificate. This parameter is ignored if the Include Signing Certificate parameter is false.

The Signing Time parameter specifies the time at which the signer purportedly performed the signing process. Leave this parameter blank to use the current time.

The Signer Roles parameter holds a newline separated list of the roles claimed by the signer. One possible way to enter this expression is to double-quote the list and use the \n escape sequence for the newline separator. To force the evaluation of the expression, surround the string literal with a call to the _concat function. For example _concat("buyer\nmanager"). The XAdES Technical Specification does not define any standard roles. A role could be something like Sales Director, which would indicate that the signer was acting as the Sales Director when he signed the document.

The TSA URL parameter is the location of the Time Stamp Authority used to create time stamps. The XAdES properties that contain time stamps are: AllDataObjectsTimeStamp, IndividualDataObjectsTimeStamp, SignatureTimeStamp, RefsOnlyTimeStamp, and SigAndRefsTimeStamp.

Signature Production Place	
City	The purported city where the signer was at the time of signature creation.

Signature Production Place	
State Or Province	The purported state or province where the signer was at the time of signature creation.
Postal Code	The purported postal code where the signer was at the time of signature creation.
Country	The purported country where the signer was at the time of signature creation.

Together, these parameters specify where the signer purportedly was at the time of signature creation. The Signature Production Place is a qualifying property of the whole signature.

Signature Policy	
Signature Policy Identifier	An Object Identifier that uniquely identifies a specific version of the signature policy. Leave this property blank to specify an Implied policy in XAdES-EPES form and above.
Signature Policy Document	Path to the file containing a copy of the Signature Policy Document. Leave this property blank to specify an Implied policy in XAdES-EPES form and above.

The policy parameters specify the Signature policy for Explicit Policy Electronic Signature forms and above. For an implied policy, simply leave both parameters empty. For an explicit policy, specify the policy ObjectIdentifer (URI or OID) and the path to the policy file. The contents of the file will be digested and the hash will appear in the SignaturePolicyIdentifier/SignaturePolicyId/SigPolicyHash element. Both parameters are ignored if the XAdES-BES form is selected.

Complete Form	
TrustStore Provider	Provider for the keystore containing the Certificate Authorities. This property is required for XAdES-C forms and above.

Complete Form	
Certificate Store Providers	Comma-separated List of Keystore, Directory CertStore or LDAP providers for the certificate stores used to retrieve revocation material. This property is required for XAdES-C forms and above.

These parameters are needed to retrieve the validation data for XAdES-C forms and above. The data is found by executing PKIX validation of the signing certificate with revocation checking enabled. The TrustStore provider specifies the keystore provider containing the Certificate Authorities to be used as trust anchors. The Certificate Store Providers parameter is a comma-separated List of providers used to retrieve revocation material.

Reference 1	
Reference 1 URI	URI to the first piece of data that will be digested and signed. If left blank, the whole XML document will be digested and signed.
Reference 1 Transform 1	First transform algorithm to apply to the first reference data.
Reference 1 Transform 1 Parameters	Parameters for the first transform algorithm to apply to the first reference data. For Exclusive Canonical XML, this is a space separated list of XML namespace prefixes. For XSLT, this is the name of a defined transform. For XPathFilter, this is an XPath expression.
Reference 1 Transform 1 XML Namespace Provider	Provider for the XML Namespace Map for XPathFilter transforms.
Reference 1 Transform 2	Second transform algorithm to apply to the first reference data.
Reference 1 Transform 2 Parameters	Parameters for the second transform algorithm to apply to the first reference data. For Exclusive Canonical XML, this is a space separated list of XML namespace prefixes. For XSLT, this is the name of a defined transform. For XPathFilter, this is an XPath expression.

Reference 1	
Reference 1 Transform 2 XML Namespace Provider	Provider for the XML Namespace Map for XPathFilter transforms.
Reference 1 MimeType	The MimeType element of the DataObjectFormat. Indicates how a user should interpret the signed data in the first reference (text, sound, video, and so on).
Reference 1 Encoding	The Encoding element of the DataObjectFormat. Indicates the encoding of the signed data in the first reference. Ignored if MimeType is left blank.
Reference 1 Description	The Description element of the DataObjectFormat. Holds textual information related to the signed data in the first reference. Ignored if MimeType is left blank.
Reference 1 Documentation URI	A DocumentationReference sub-element of the ObjectIdentifier element of the DataObjectFormat. Points to a document where additional information about the nature of the data object can be found. Ignored if MimeType is left blank.
Reference 1 Identifier	The Identifier sub-element of the ObjectIdentifier element of the DataObjectFormat. Contains a permanent identifier of the nature of the object. Ignored if MimeType is left blank.
Reference 1 Commitment	Commitment type that applies to this signed data object.
Reference 1 Commitment Description	The text description for the commitment type that applies to this signed data object. A default English description will be used if a standard commitment type is chosen and this property is left blank.
Reference 1 Time Stamp	Requests a time stamp to be computed before the signature production, over a sequence formed by some of the ds:Reference elements within the ds:SignedInfo referencing whatever the signer wants to sign except the SignedProperties element.

The reference URIs supported are: <empty string> for the whole XML document; #idattrib for the same-document sub-tree rooted at the element that has an ID attribute with value idattrib; http://host:port/page for the resource located at this HTTP address, and possibly other URLs supported by the library.

The Reference 1 URI parameter is the URI to the first piece of data that will be digested and signed. If left blank, the whole XML document will be digested and signed.

The Reference 1 Transform 1 is the first transform algorithm to apply to the reference data. The Reference 1 Transform 1 Parameters contain the parameters for the transform. Similarly, the Reference 1 Transform 2 is the second transform and Reference 1 Transform 2 Parameters specify its parameters.

For more information on the transforms, see the table in this section that lists and describes the transforms available to the digital signature service.

The remaining parameters in this group are reference-specific qualifying properties.

The MimeType, Encoding, Description, and Documentation URI parameters together form the contents of the DataObjectFormat for this particular reference.

The Commitment and Commitment Description parameters are similar to the All Signed Data Objects Commitment and All Signed Data Objects Commitment Description parameters, except they apply to a single reference. Refer to the table on the Qualifying Properties group earlier in this section for an explanation of the commitment types in the XAdES Technical Specification.

The Time Stamp boolean parameter indicates whether an IndividualDataObjectsTimeStamp element is generated for this reference.

Subsequent references (2, 3) are similar to reference 1 except a missing reference URI indicates the end of the list of references instead of the whole document.

The list of transforms per reference is not limited to 2. Any number of transforms can be specified using user parameters.

The list of references is not limited to 2. Any number of references can be specified using user parameters.

Reference 2	
Reference 2 URI	URI to the second piece of data that will be digested and signed. If you need more references, create user parameters named ref[X]uri, ref[X]transform[Y], ref[X]transform[Y]parms, ref[X]transform[Y]nsmap, ref[X]formatmime, ref[X]formatenc, ref[X]formatdesc, ref[X]formatdocuri, ref[X]formatident, ref[X]commitment, ref[X]timestamp, where X >= 3, Y >= 1. For example, ref3transform2 is the second transform of the third reference.
Reference 2 Transform 1	First transform algorithm to apply to the second reference data.
Reference 2 Transform 1 Parameters	Parameters for the first transform algorithm to apply to the second reference data. For Exclusive Canonical XML, this is a space separated list of XML namespace prefixes. For XSLT, this is the name of a defined transform. For XPathFilter, this is an XPath expression.
Reference 2 Transform 1 XML Namespace Provider	Provider for the XML Namespace Map for XPathFilter transforms.
Reference 2 Transform 2	Second transform algorithm to apply to the second reference data.
Reference 2 Transform 2 Parameters	Parameters for the second transform algorithm to apply to the second reference data. For Exclusive Canonical XML, this is a space separated list of XML namespace prefixes. For XSLT, this is the name of a defined transform. For XPathFilter, this is an XPath expression.
Reference 2 Transform 2 XML Namespace Provider	Provider for the XML Namespace Map for XPathFilter transforms.

Reference 2	
Reference 2 MimeType	The MimeType element of the DataObjectFormat. Indicates how a user should interpret the signed data in the second reference (text, sound, video, and so on).
Reference 2 Encoding	The Encoding element of the DataObjectFormat. Indicates the encoding of the signed data in the second reference. Ignored if MimeType is left blank.
Reference 2 Description	The Description element of the DataObjectFormat. Holds textual information related to the signed data in the second reference. Ignored if MimeType is left blank.
Reference 2 Documentation URI	A DocumentationReference sub-element of the ObjectIdentifier element of the DataObjectFormat. Points to a document where additional information about the nature of the data object can be found. Ignored if MimeType is left blank.
Reference 2 Identifier	The Identifier sub-element of the ObjectIdentifier element of the DataObjectFormat. Contains a permanent identifier of the nature of the object. Ignored if MimeType is left blank.
Reference 2 Commitment	Commitment type that applies to this signed data object.
Reference 2 Commitment Description	The text description for the commitment type that applies to this signed data object. A default English description will be used if a standard commitment type is chosen and this property is left blank.
Reference 2 Time Stamp	Requests a time stamp to be computed before the signature production, over a sequence formed by some of the ds:Reference elements within the ds:SignedInfo referencing whatever the signer wants to sign except the SignedProperties element.

The Reference 2 parameters are similar to the Reference 1 parameters. Refer to the Reference 1 group above for details.

The following table lists the transforms available. Some transforms have implicit parameters and do not require any explicit parameters. Other transforms take parameters, as described in the following table.

Transforms Available to Digital Signature Service	
Base64 *http://www.w3.org/2000/09/xmldsig#base64*	This transform decodes the Base64 encoded character data. If the input is a node-set, then the string-value of the node-set is decoded (ignoring the element tags, comments and processing instructions). This transform takes no explicit parameters.
Enveloped Signature *http://www.w3.org/2000/09/xmldsig#enveloped-signature*	This transform removes the Signature element from the calculation of the signature when the signature is within the content that it is being signed. This transform takes no explicit parameters.
Exclusive Canonical XML *http://www.w3.org/2001/10/xml-exc-c14n#*	This transform is useful when message parts can be enveloped and stripped off to construct new messages. Exclusive Canonical XML ignores the namespace context inherited from parent elements. This keeps the digested data constant despite these operations. This transform takes an optional space-separated list of XML namespace prefixes declared in the XML Namespace provider. These are additional prefixes to be ignored.
Exclusive Canonical XML With Comments *http://www.w3.org/2001/10/xml-exc-c14n#WithComments*	This transform is similar to Exclusive Canonical XML except comments are preserved in the digested data. This transform takes an optional space-separated list of XML namespace prefixes declared in the XML Namespace provider. These are additional prefixes to be ignored.

Transforms Available to Digital Signature Service	
XPathFilter *http://www.w3.org/TR/1999/REC-xpath-19991116*	This transform evaluates the XPath expression for each node in the input node-set and keeps only the nodes where the expression evaluated to true. This transform takes the XPath expression in ref[X]transform[Y]parms1 and optionally an XML Namespace provider name in ref[X]transform[Y]parms1nsmap to declare a namespace map.
XSLTTransform *http://www.w3.org/TR/1999/REC-xslt-19991116*	This transform indicates an XSLT stylesheet must be used and the result is what is referenced for signing. This transform takes the name of a defined transform as parameter (similar to what is done with the XDGenTransform service). The defined transform must be an XSLT transform and return XML.
Inclusive Canonical XML 1.0 *http://www.w3.org/TR/2001/REC-xml-c14n-20010315*	This transform performs typical XML Canonicalization that attracts the xml namespace declarations from the inherited context. This canonicalization is the default if the last transform returns a node-set. This transform takes no parameters.
Inclusive Canonical XML With Comments 1.0 *http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments*	This transform is similar to Inclusive Canonical XML except comments are preserved. This transform takes no parameters.

Transforms Available to Digital Signature Service	
Inclusive Canonical XML 1.1 *http://www.w3.org/2006/12/xml-c14n11*	Canonical XML 1.1 is a revision to Canonical XML 1.0 to address issues related to inheritance of attributes in the XML namespace when canonicalizing document subsets, including the requirement not to inherit xml:id, and to treat xml:base URI path processing properly. This canonicalization is a better choice than the default Inclusive Canonical XML 1.0. This transform takes no parameters.
Inclusive Canonical XML With Comments 1.1 *http://www.w3.org/2006/12/xml-c14n11#WithComments*	This transform is similar to Inclusive Canonical XML 1.1 except comments are preserved. This transform takes no parameters.

ref[X]transform[Y]parms where X >= 3 and Y >= 1, for example, ref3transform2 is the second transform of the third reference.

When the Tree level is selected in the trace settings, the service will log the referenced data that was actually digested.

Edges:

The following table lists and describes the edges that are returned by the XML Digital Signature Create service.

Edge	Description
success	The Signature was successfully inserted.
fail_parse	An iFL or XPath expression could not be evaluated.
fail_operation	The Signature could not be inserted.

Examples

For more information on related examples of XML Digital Signatures, see *XML Digital Signature Create Service* on page 98. In particular, *Example 2: Simple SOAP Message* on page 112 shows how an XPath expression for the signature parent element can instruct the service to find or construct that path. There are also examples of Transform with Transform parameters.

Note: The examples in this section are specific to the XAdES Digital Signature Create service (com.ibi.agents.XDXAdESCreateAgent). For your convenience, the sample input and output documents are attached to the PDF in unabbreviated and unindented form.

For PDF-compatibility purposes, the file extension of the *XAdESCreate.zip* file is temporarily renamed to *.zap*. After saving this file to your file system, you must rename this extension back to *.zip* before the file can be used.

Example 1: Enveloped Basic Electronic Signature

The XAdES Digital Signature Create service has a large number of parameters but very few are actually required. At a minimum, the signing key must be specified using the KeyStore Provider and Signing Key Alias parameters. The Signing Key Password must also be specified if it is different than the KeyStore password. Everything else is optional. Since we use the default empty reference URI to sign the whole document, we must also specify the Enveloped Signature transform.

This table lists the parameter values for this example. Other parameters that are not listed have their default value.

Parameter	Value
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
Reference 1 Transform 1	http://www.w3.org/2000/09/xmldsig#enveloped-signature

A sample input document is shown as follows (indented for display purposes only):

```
<Message>
   <Body>...</Body>
</Message>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Body>...</Body>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="" Id="xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556-ref0">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>2HxElKLdnrWUim5oPiaRdHKlyJ2fQdDckdiaboWT7Vl=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556-signedprops" Type="http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>2GHN6Ndk5m25CcQLoLCbF/j9qiBTQHqoWxD9cFlab3c=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556-sigvalue">
i8OgEjg9dZpq2PsuDHrg9SpEizBvR/Q4iJH/e/xca1HOVo+ZMKz1Lu2nbCLh9A2YR0/TLRP3jMarrXz2QcMaMx2CxM6HhwFoMlpTifNMCQrHY5iOhJ4hcfBCWlgymCXKdiKr
YaYcsjjCWel/IJSqQ2V/8kALdXXmuEIP05AssWl=</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM...</ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="#xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556" xmlns:xades141="
http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIlkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSwI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
               </xades:SignedSignatureProperties>
            </xades:SignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

The signature is appended to the end of the parent element. In this case, the default parent is the root element. This explains why the Body appears before the Signature. The Signature contains two references. The first reference was configured in the service to cover the whole document except the signature itself. The second reference was added automatically to cover the SignedProperties.

A XAdES Signature is a regular XML Digital Signature with extra properties. Those properties appear in a ds:Object element within the signature. The QualifyingProperties contain the SIgnedProperties and the UnsignedProperties. In this simple case, there are no UnsignedProperties.

The SigningTime is the first property under the QualifyingProperties element. The service picked the current time since the Signing Time parameter was left blank.

Multiple certificates can bind the same private key to multiple identities. XAdES dictates the Signing Certificate must be unambiguously declared to show in which capacity the signer signed the document. Here the Signing Certificate appears in the KeyInfo. The KeyInfo is not signed, but a hash of the Signing Certificate also appears under the SignedProperties. The SigningCertificate holds a reference to each certificate in the signer certificate chain. The Issuer and Serial Number pair plus a hash uniquely identify each certificate.

Example 2: Optional Qualifying Properties

XAdES has many optional Qualifying Properties. This example shows how to add more qualifying properties to the signature.

This table lists the parameter values for this example. Other parameters that are not listed have their default value.

Parameter	Value
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
All Signed Data Objects Commitment	http://uri.etsi.org/01903/v1.2.2#ProofOfOrigin
All Signed Data Objects Time Stamp	true
Sign Signing Certificate	true
Signer Roles	_concat("Buyer\nSales Director")

Parameter	Value
City	New York
State Or Province	NY
Postal Code	10121
Country	US
Reference 1 URI	#myid

A sample input document is shown as follows (indented for display purposes only):

```
<Message>
   <Body Id="myid">...</Body>
</Message>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Body Id="myid">...</Body>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-7c99ef41-239c-44f5-adb6-bc88862aabc3">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="#myid" Id="xmldsig-7c99ef41-239c-44f5-adb6-bc88862aabc3-ref0">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>e95XCAqcyE0xHBhZVSTL1zShbebjcvHB+7LVt14rOuo=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-7c99ef41-239c-44f5-adb6-bc88862aabc3-signedprops" Type="http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>7Xqf4rHahYT/8XGYaVW19pqA6jK3PiEYYwoQqxmFxv4=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-7c99ef41-239c-44f5-adb6-bc88862aabc3-keyinfo">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>86jQDv0crUINBFRsCXNrQi02JFpMzV5ZvdT9K+2TtGc=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-7c99ef41-239c-44f5-adb6-bc88862aabc3-sigvalue">
YA2Eh8+18tJ/IPPFcTxlpGsXZtbJVmxzBF6P/v3RNdEw9W+dA9Jk6G7D39hl4bblfClAwRmhfMGv8bJ9A3MpGr74Oo7ZXRVPfTwD4NWywtVCu2hXpQbGjV447UBXswWb
1Sz1Ffs8R0HOl+ITvmGlAycZlUF7VZHwoz3Na+pyao=</ds:SignatureValue>
      <ds:KeyInfo Id="xmldsig-7c99ef41-239c-44f5-adb6-bc88862aabc3-keyinfo">
         <ds:X509Data>
            <ds:X509Certificate>MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM...</ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="#xmldsig-7c99ef41-239c-44f5-adb6-bc88862aabc3" xmlns:xades141="
http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-7c99ef41-239c-44f5-adb6-bc88862aabc3-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIlkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
                  <xades:SignatureProductionPlace>
                     <xades:City>New York</xades:City>
                     <xades:StateOrProvince>NY</xades:StateOrProvince>
                     <xades:PostalCode>10121</xades:PostalCode>
                     <xades:CountryName>US</xades:CountryName>
                  </xades:SignatureProductionPlace>
                  <xades:SignerRole>
                     <xades:ClaimedRoles>
                        <xades:ClaimedRole>Sales Director</xades:ClaimedRole>
                        <xades:ClaimedRole>Buyer</xades:ClaimedRole>
                     </xades:ClaimedRoles>
                  </xades:SignerRole>
               </xades:SignedSignatureProperties>
               <xades:SignedDataObjectProperties>
                  <xades:CommitmentTypeIndication>
                     <xades:CommitmentTypeId>
                        <xades:Identifier>http://uri.etsi.org/01903/v1.2.2#ProofOfOrigin</xades:Identifier>
                        <xades:Description>Indicates that the signer recognizes to have created, approved and sent the signed data object</xades:Description>
                     </xades:CommitmentTypeId>
                     <xades:AllSignedDataObjects/>
                  </xades:CommitmentTypeIndication>
                  <xades:AllDataObjectsTimeStamp>
                     <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                     <xades:EncapsulatedTimeStamp>MIIPPwYJKoZIhvcNAQcCoIIPMDCCDywCAQMxCzAJBgUrDgMCGgUAMIG7BgsqhkiG9w0BCRABBKCB...
                     </xades:EncapsulatedTimeStamp>
                  </xades:AllDataObjectsTimeStamp>
               </xades:SignedDataObjectProperties>
            </xades:SignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

This signature has three references: the first reference is declared in the service, the second reference covers the SignedProperties, and the third reference covers the KeyInfo because we asked to sign the Signing Certificate. The last two references were added automatically by the service.

The QualifyingProperties element contains more properties. The SigningTime and SigningCertificate are familiar from example 1. All the other properties are new in example 2.

The City, State Or Province, Postal Code, and Country parameters combine to form the SignatureProductionPlace property.

The SignerRole property lists two ClaimedRoles: Buyer and Sales Director.

The signer declares he is the originator of this message by claiming the Proof of Origin commitment. This commitment applies to all references because of the presence of the AllSignedDataObjects element. Since the All Signed Data Objects Commitment Description parameter is left blank, a default commitment description appears in the signature.

The AllDataObjectsTimeStamp is a time stamp calculated over all the references except the one marked with Type attribute equal to "http://uri.etsi.org/01903#SignedProperties".

Example 3: Implied Policy

To specify the Signature Policy, the XAdES form must be EPES or above. The XAdES Technical Specification states:

A signature policy is useful to clarify the precise role and commitments that the signer intends to assume with respect to the signed data object, and to avoid claims by the verifier that a different signature policy was implied by the signer.

The signer may reference the policy either implicitly or explicitly. An implied policy means the signer follows the rules of the policy but the signature does not indicate which policy. It is assumed the choice of policy is clear from the context in which the signature is used. When the policy is not implied, the signature contains an ObjectIdentier (URI or OID) that uniquely identifies the version of the policy in use. The signature also contains a hash of the policy document to make sure the signer and verifier agree on the contents of the policy document.

Example 3 demonstrates an implied policy. This is obtained by setting the XAdES form to EPES and leaving the Signature Policy Identifier and Signature Policy Document parameters blank.

This table lists the parameter values for this example. Other parameters that are not listed have their default value.

Parameter	Value
XAdES Form	XAdES-EPES
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
Reference 1 URI	#myid

A sample input document is shown as follows (indented for display purposes only):

```
<Message>
   <Body Id="myid">...</Body>
</Message>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Body Id="myid">...</Body>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="#myid" Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-ref0">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>e95XCAqcyE0xHBhZVSTL1zShbebjcvHB+7LVt14rOuo=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-signedprops" Type="http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>krFeNaF03HOBTKeBVwXnBLjqUXyschFIFLOTpttwszY=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-sigvalue">
N2HU1hIMBXDddBWMsyXdaqDbZFpePd6xGAYbPQ8nT9771ycneqhaSxQ26isKz/vYxbEqbG9CbE73RPQsWelt4vDuXvxYulmmwWxRaOvnlhvL6mvFp
VqyjFBlftyCxgGIKn59w9zzNJ4dLyxxD3nanNrEQaU3ENd1+PJEPql0T7Q=</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM...</
ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="#xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88"
xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIlkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
                  <xades:SignaturePolicyIdentifier>
                     <xades:SignaturePolicyImplied/>
                  </xades:SignaturePolicyIdentifier>
               </xades:SignedSignatureProperties>
            </xades:SignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

The property to notice is the SignaturePolicyIdentifier with the SignaturePolicyImplied empty element.

Example 4: Explicit Policy Identifier

This example demonstrates an explicit policy identifier. This is obtained by setting the XAdES form to EPES, and assigning values to the two policy parameters. The Signature Policy Identifier is a URI or OID that uniquely identifies the version of the policy document. The Signature Policy Document is the path to the policy file in the file system. The signature will contain a hash of the policy to prove both the signer and verifier agree on the contents of the policy. It is important to keep the policy file intact in order to keep the hash constant. It would be wise to make the policy file read-only.

This table lists the parameter values for this example. Other parameters that are not listed have their default value.

Parameter	Value
XAdES Form	XAdES-EPES
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
Signature Policy Identifier	http://iwaysoftware.com/xades#policy1.0
Signature Policy Document	policy-1.0.doc
Reference 1 URI	#myid

A sample input document is shown as follows (indented for display purposes only):

```
<Message>
   <Body Id="myid">...</Body>
</Message>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Body Id="myid">...</Body>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-0dd226c3-3567-4e9a-aedc-df6b4d8b1e41">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="#myid" Id="xmldsig-0dd226c3-3567-4e9a-aedc-df6b4d8b1e41-ref0">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>e95XCAqcyE0xHBhZVSTL1zShbebjcvHB+7LVt14rOuo=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-0dd226c3-3567-4e9a-aedc-df6b4d8b1e41-signedprops" Type="http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>FPTRCc4DkRg4qaWSKkzqaKOHwAR5Cb8OSYWzBcnmFPo=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-0dd226c3-3567-4e9a-aedc-df6b4d8b1e41-sigvalue">
LAOnykVp6nWPQEot0tszn+SwqEl3226/NjHSISv2sVvMEqjj1p4yEi0wFwyXgbh5/MoOFR4rX96O96IYCApL8mRCHl/hgqOlfMBL4ck8/ARMP5JbYeo5+J
HkZcD495OWNDnW2RAp9uTJLzKwJLZhSr7k4U4TJFUeLvuy6o2LiB4=</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM...</
ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="#xmldsig-0dd226c3-3567-4e9a-aedc-df6b4d8b1e41"
xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-0dd226c3-3567-4e9a-aedc-df6b4d8b1e41-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
                  <xades:SignaturePolicyIdentifier>
                     <xades:SignaturePolicyId>
                        <xades:SigPolicyId>
                           <xades:Identifier>http://iwaysoftware.com/xades#policy1.0</xades:Identifier>
                        </xades:SigPolicyId>
                        <xades:SigPolicyHash>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>NEze6znOB+FKA0yaRrav1wmWnCnsWhC4vnL//dAgMEs=</ds:DigestValue>
                        </xades:SigPolicyHash>
                     </xades:SignaturePolicyId>
                  </xades:SignaturePolicyIdentifier>
               </xades:SignedSignatureProperties>
            </xades:SignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

The property to notice is the SignaturePolicyIdentifier with a SigPolicyId and a SigPolicyHash.

Example 5: Reference Specific Properties

This example shows the effect of qualifying properties that pertain to a specific reference. Two references are declared with different qualifying properties. Contrast this with *Example 2: Optional Qualifying Properties* on page 169 where the qualifying properties applied to the signature itself or all the references at once.

This table lists the parameter values for this example. Other parameters that are not listed have their default value.

Parameter	Value
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
Reference 1 URI	#id1
Reference 1 MimeType	audio/mpeg
Reference 1 Encoding	base64
Reference 1 Description	MP3 file encoded in base64
Reference 1 Documentation URI	http://iwaysoftware.com/xades/audio.html
Reference 1 Identifier	http://iwaysoftware.com/xades#mp3
Reference 2 URI	#id2
Reference 2 Commitment	*http://uri.etsi.org/01903/v1.2.2#ProofOfApproval*
Reference 2 Commitment Description	Signer has approved the content
Reference 2 Time Stamp	true

A sample input document is shown as follows (indented for display purposes only):

```
<Message>
   <Part1 Id="id1">...</Part1>
   <Part2 Id="id2">...</Part2>
</Message>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Part1 Id="id1">...</Part1>
   <Part2 Id="id2">...</Part2>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="#id2" Id="xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6-ref0">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>ZmEjfz6t79qAaOluz4oeCsgDGEA63r4n7ae/l/LurOI=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#id1" Id="xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6-ref1">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>0UYW8dZZfz4S6Imw29e0deikHStWsQFQwOkGYDCkYkA=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6-signedprops" Type="http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>OERtd9vC4DWVL5uk0rgc0Y6yy4jKgzDiYIXlxNAVAiY=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6-sigvalue">
OmGiojR/ZetDuwzKMggULW3c+qzaSmgwYLNXCHLECGT6+/gwXiWfUHsM6GLzcRT8puL7ttu5tOUy5Sv3Nlts1TtngMzfUlMUgoCdzmSpmnO58P3FTt2fC6
088vnZDdolUmXWUIBV2LcIaJpVMCIDlbWoqLWpY2+hqR4nyjFX7BE=</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM...</
ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="#xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6"
xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIlkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
               </xades:SignedSignatureProperties>
               <xades:SignedDataObjectProperties>
                  <xades:DataObjectFormat ObjectReference="#xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6-ref1">
                     <xades:Description>MP3 file encoded in base64</xades:Description>
                     <xades:ObjectIdentifier>
                        <xades:Identifier>http://iwaysoftware.com/xades#mp3</xades:Identifier>
                        <xades:DocumentationReferences>
                           <xades:DocumentationReference>http://iwaysoftware.com/xades/audio.html</xades:DocumentationReference>
                        </xades:DocumentationReferences>
                     </xades:ObjectIdentifier>
                     <xades:MimeType>audio/mpeg</xades:MimeType>
                     <xades:Encoding>base64</xades:Encoding>
                  </xades:DataObjectFormat>
                  <xades:CommitmentTypeIndication>
                     <xades:CommitmentTypeId>
                        <xades:Identifier>http://uri.etsi.org/01903/v1.2.2#ProofOfApproval</xades:Identifier>
                        <xades:Description>Signer has approved the content</xades:Description>
                     </xades:CommitmentTypeId>
                     <xades:ObjectReference>#xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6-ref0</xades:ObjectReference>
                  </xades:CommitmentTypeIndication>
                  <xades:IndividualDataObjectsTimeStamp>
                     <xades:Include referencedData="true" URI="#xmldsig-241a9969-8955-49e7-8938-13e4d3f313a6-ref1"/>
                     <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                     <xades:EncapsulatedTimeStamp>
MIIPPwYJKoZIhvcNAQcCoIIPMDCCDywCAQMxCzAJBgUrDgMCGgUAMIIG7BgsqhkiG9w0BCRABBKCB...</xades:EncapsulatedTimeStamp>
                  </xades:IndividualDataObjectsTimeStamp>
               </xades:SignedDataObjectProperties>
            </xades:SignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

The signature contains three references: two configured in the service and one added automatically for the SignedProperties. The Reference specific qualifying properties are found within the SignedDataObjectProperties element. The DataObjectFormat property qualifies the first reference as can be seen by the URI in the ObjectReference attribute. The CommitmentTypeIndication qualifies the second reference as can be seen by the URI in the ObjectReference element. The commitment is described by a custom commitment description. Finally, the IndividualDataObjectsTimeStamp contains a type stamp for the second reference as can be seen in the URI attribute.

Example 6: Electronic Signature With Time

This example demonstrates the XAdES-T form.

This table lists the parameter values for this example. Other parameters that are not listed have their default value.

Parameter	Value
XAdES Form	XAdES-T
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
Reference 1 URI	#myid

A sample input document is shown as follows (indented for display purposes only):

```
<Message>
   <Body Id="myid">...</Body>
</Message>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Body Id="myid">...</Body>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-98490575-75a7-48e8-a3bb-5c3c666f09ef">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sa-sha1"/>
         <ds:Reference URI="#myid" Id="xmldsig-98490575-75a7-48e8-a3bb-5c3c666f09ef-ref0">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>e95XCAqcyE0xHBhZVSTL1zShbebjcvHB+7LVt14rOuo=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-98490575-75a7-48e8-a3bb-5c3c666f09ef-signedprops" Type="
http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>WUxonT68FI75KuMeBBTlHiGE/s+uGC3VgDXFyTI+V14=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-98490575-75a7-48e8-a3bb-5c3c666f09ef-sigvalue">
cwsXmtbcTKWqMo7VlvN0tVlSrdTo505nZbluNWhozQ7cRW1uedsYE8hfx8lkK3rLe8UOg8Yr+bxPmqz9/ICska2DdWqF4MF1wzCJRkoQk
+SYQsZ1ae9et3tkVaDZ/Gn/rrLKCEizCmDVmQVyUAlnocrTvSZZSUthxddg5ZYAFo8=</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>
MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM...</ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="
#xmldsig-98490575-75a7-48e8-a3bb-5c3c666f09ef" xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-98490575-75a7-48e8-a3bb-5c3c666f09ef-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIlkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
                  <xades:SignaturePolicyIdentifier>
                     <xades:SignaturePolicyImplied/>
                  </xades:SignaturePolicyIdentifier>
               </xades:SignedSignatureProperties>
            </xades:SignedProperties>
            <xades:UnsignedProperties>
               <xades:UnsignedSignatureProperties>
                  <xades:SignatureTimeStamp>
                     <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                     <xades:EncapsulatedTimeStamp>
MIIPPwYJKoZIhvcNAQcCoIIPMDCCDywCAQMxCzAJBgUrDgMCGgUAMIG7BgsqhkiG9w0BCRABBKCB...</
xades:EncapsulatedTimeStamp>
                  </xades:SignatureTimeStamp>
               </xades:UnsignedSignatureProperties>
            </xades:UnsignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

The XAdES-T form is a superset of the XAdES-EPES form. Since the Signature Policy Identifier and the Signature Policy Document parameters are unspecified, this produces an implied policy.

The SignatureTimeStamp mandated by the XAdES-T form appears as an unsigned property within the QualifyingProperties.

Example 7: Complete Validation Data References

This example demonstrates the XAdES-C form.

This table lists the parameter values for this example. Other parameters that are not listed have their default value.

Parameter	Value
XAdES Form	XAdES-C
KeyStore Provider	ksprov
Signing Key Alias	alias1
Signing Key Password	secret
Signing Key Password	secret
Signing Key Password	secret
Reference 1 URI	#myid

A sample input document is shown as follows (indented for display purposes only):

```
<Message>
   <Body Id="myid">...</Body>
</Message>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Body Id="myid">...</Body>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-b57aca9b-3bff-49ba-88a1-56b4455e4a09">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="#myid" Id="xmldsig-b57aca9b-3bff-49ba-88a1-56b4455e4a09-ref0">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>e95XCAqcyE0xHBhZVSTL1zShbebjcvHB+7LVt14rOuo=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-b57aca9b-3bff-49ba-88a1-56b4455e4a09-signedprops" Type="
http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>DpuL6OlXa/OgLhN0E24kdeHZ7hJfqvzYuHr5iQdYKAQ=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-b57aca9b-3bff-49ba-88a1-56b4455e4a09-sigvalue">
XfZsD42wXCxNldTCYyci8OQtHUXkLXmxegNOa3C+YI5AJY/CmLY7J0T4TnlzG28GSmE4rhkoXURZ4R9vx7JlJ2eRukDongpufqmHwrMPl4
bP+FkF73pDT9HKs1vDhBmo4/3dwon+SAhegwyEnLDY1D6CPSNlcvrzRgvh6Y9GxaE=</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>
MIICejCCAaOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM...</ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="
#xmldsig-b57aca9b-3bff-49ba-88a1-56b4455e4a09" xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-b57aca9b-3bff-49ba-88a1-56b4455e4a09-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIlkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
                  <xades:SignaturePolicyIdentifier>
                     <xades:SignaturePolicyImplied/>
                  </xades:SignaturePolicyIdentifier>
               </xades:SignedSignatureProperties>
            </xades:SignedProperties>
            <xades:UnsignedProperties>
               <xades:UnsignedSignatureProperties>
                  <xades:SignatureTimeStamp>
                     <ds:CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
                     <xades:EncapsulatedTimeStamp>
MIIPPwYJKoZIhvcNAQcCoIIPMDCCDywCAQMxCzAJBgUrDgMCGgUAMIG7BgsqhkiG9w0BCRABBIGkCB...</
xades:EncapsulatedTimeStamp>
                  </xades:SignatureTimeStamp>
                  <xades:CompleteCertificateRefs>
                     <xades:CertRefs>
                        <xades:Cert>
                           <xades:CertDigest>
                              <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                              <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                           </xades:CertDigest>
                           <xades:IssuerSerial>
                              <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                              <ds:X509SerialNumber>15</ds:X509SerialNumber>
                           </xades:IssuerSerial>
                        </xades:Cert>
                     </xades:CertRefs>
                  </xades:CompleteCertificateRefs>
                  <xades:CompleteRevocationRefs>
                     <xades:CRLRefs>
                        <xades:CRLRef>
                           <xades:DigestAlgAndValue>
                              <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                              <ds:DigestValue>UBNPGGiTIRFHjNG4r+OAfYRrgSlvYI8HR205k4Lxy+w=</ds:DigestValue>
                           </xades:DigestAlgAndValue>
                           <xades:CRLIdentifier>
                              <xades:Issuer>CN=Example CA,O=iWay,C=US</xades:Issuer>
                              <xades:IssueTime>2011-11-21T10:44:33.000-05:00</xades:IssueTime>
                              <xades:Number>17230332170</xades:Number>
                           </xades:CRLIdentifier>
                        </xades:CRLRef>
                     </xades:CRLRefs>
                  </xades:CompleteRevocationRefs>
               </xades:UnsignedSignatureProperties>
            </xades:UnsignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

The XAdES-C form is a superset of the XAdES-T form. Therefore, the signature contains an implied policy and a SignatureTimeStamp, like *Example 6: Electronic Signature With Time* on page 180.

XAdES Digital Signature Verify Service

Syntax:

`com.ibi.agents.XDXAdESVerifyAgent`

Description:

This service is used to validate an XML Advanced Electronic Signature. Refer to the XAdES Digital Signature Create service for some background information on XAdES. This guide assumes the reader is familiar with the XAdES specification.

Parameters:

The following tables describe the parameters for the XAdES Digital Signature Verify service. Each table is followed by a discussion of that parameter group.

Signature Location	
XML Namespace Provider	Provider for the mapping between XML namespace prefix and namespace URI. If left blank, the XPath expression in the Element Path and Required Signature Coverage parameters cannot contain namespaces.
XPath Syntax	Determines which syntax level of XPath should be used. You can select the iWay abbreviated syntax or the XPath 1.0 full syntax. The default option selects the syntax level as set in the General Settings area of the iSM Administration Console.
Signature Element Path	Path to the signature XML element. If left blank, the service will search throughout the document for an element named Signature in the namespace http://www.w3.org/2000/09/xmldsig#.
Remove Security Parent Element	If set to True, the WSSE Security parent element is removed from the document after the verification is successful.

The Signature Element Path parameter is an XPath expression that evaluates to the Signature node. For example, for a SOAP message, the XPath expression would be a path somewhere within the SOAP Headers. The XPath expression can be the union of two paths if the application is expecting signatures in one of two locations. An XML Namespace Provider is needed if the XPath expression contains namespaces.

The Remove Security Parent Element boolean parameter is a way to remove a SOAP header after it has been processed.

Signature Criteria	
TrustStore Provider	Provider for the keystore containing the Certificate Authorities.
Certificate Store Providers	Comma-separated List of Keystore, Directory CertStore or LDAP providers for the certificate stores used to complete the certificate chain and to retrieve revocation material.
Enable Certificate Revocation	If set to true, use the CRLs from the CertStores to check whether the certificate of the signer has been revoked.
Maximum Path Length	Maximum number of non-self-issued intermediate certificates that may exist in a certification path. The last certificate in a certification path is not included in this limit. 0 implies that the path can only contain a single certificate. -1 implies that there is no maximum. If any of the CA certificates contain the BasicConstraintsExtension, the value of the pathLenConstraint field of the extension overrides the Maximum Path Length parameter.
Enforce KeyUsage Extension	Determines how the KeyUsage Extension is verified when present in the signer certificate. The options are Not Enforced, digitalSignature, nonRepudiation, digitalSignature OR nonRepudiation, digitalSignature AND nonRepudiation.

Signature Criteria	
Message Digest JCE Provider	JCE Provider for the MessageDigest service. If left blank, the default JCE provider for the MessageDigest service will be used.
Required Signature Coverage	An XPath expression that returns a node-set, where each node in the set must have been signed by the Signature to be considered valid.
Unsigned Attachment	Action to perform when a document contains an unsigned attachment. Select one of the following values from the drop-down list: ❑ Keep Unsigned Attachment {keep} (default) ❑ Remove Unsigned Attachment {remove} ❑ Fail Validation {fail}

The TrustStore parameter gives the name of the KeyStore provider that declares a KeyStore file containing the certificates of the Trusted CAs. It is highly recommended to use this KeyStore exclusively as a TrustStore. It should not contain private keys or any other unrelated certificates. The JRE includes a sample TrustStore, which is located in the following directory:

```
<java-home>/lib/security/cacerts
```

This file is convenient to retrieve the certificates of some well known CAs. It is not recommended to use the JRE sample TrustStore directly. It is better to restrict the contents of the TrustStore to the small list of Trusted CAs the application is willing to trust.

A signature can be valid and still be useless. The application must also verify that all sensitive information in the document has been covered by the signature. Otherwise, the information could be modified after the fact without affecting the signature. The service can verify the signature has appropriate coverage.

The Required Signature Coverage parameter is an XPath expression that returns a NodeSet. Each node in the NodeSet must have been signed to consider the signature valid. For every sensitive node, the XPath expression should return that node or one of its ancestors. Notice that the parameter does not demand the presence of a node. If a node is missing in the document, then it does not matter whether that node has been signed or not because there is no possibility the application could use that node anyway. The application can reject the incomplete message at a later time, but that is unrelated to the signature.

For example, if the application expects a SOAP message with a BinarySecurityToken in a SOAP Header, it can specify the union of the path to the security token and the SOAP Body in the XPath expression. Since a Signature never covers itself, the Signature element should never be returned by the XPath expression.

To avoid the dual evaluation of the transforms, that feature assumes the transforms are hierarchical starting at the node pointed to by the Reference URI. The inclusive and exclusive XML canonical transforms are supported, but the XPath Filter and XSLT transforms are not.

If the application accesses the attachments, then it should ensure they have been covered by the signature. The Unsigned Attachment parameter determines what action to perform when there are unsigned attachments. The service can keep the unsigned attachments and accept the signature, remove the unsigned attachments and accept the signature, or fail validation. There is no option to demand the presence of an attachment since there is no possibility the application could use an unsigned attachment if that attachment is absent. Again, the application can reject incomplete messages at a later time.

XAdES Criteria	
Minimum XAdES Form	Specifies the minimum acceptable XAdES form. Signatures simpler than this form will cause a validation failure.
Signature Policy Provider	The name of a Signature Policy Provider. This specifies the mapping from ObjectIdentifier (URI or OID) to policy file. The policy file is needed to verify signatures with an explicit policy. If left blank, only absent or implied policies can be validated.
Accept Implied Policy	Indicates whether the verifier should accept an implied signature policy.
Accept Unknown Properties	Indicates whether the verifier should accept unknown properties. This only affects the unsigned properties.

A XAdES signature is a regular XML Digital Signature with extra signed and unsigned properties. The specification is organized in a handful of signature forms that define strictly increasing set of properties from the simplest form to the most complex one.

The Minimum XAdES Form parameter is a convenient way to require some qualifying properties be present in the signature. The XAdES-BES form is the simplest form, therefore this value accepts every XAdES signature. The XAdES-EPES form demands the presence of a signature policy. The XAdES-T form requires the XAdES-EPES properties plus the presence of a Signature time stamp. Finally, the XAdES-C form demands the XAdES-T properties plus the complete certificate and revocation references.

A signature policy is a document in human readable form. It is an agreement between the signer and verifier describing the rules to follow to produce and verify the signature, and what claims can be made given the rules are satisfied. For example, the policy may demand a certain signing key size, or that certification revocation must be checked. The policy may say the signer agrees to consider his electronic signature legally binding in the context of sending purchase orders to his supplier, provided the policy rules are followed, of course.

The Policy can be absent, implied or explicit in the Signature. An absent policy is not mentioned at all, so the verifier can make no assumptions whether a policy is effectively in use. An implied policy specifies that a policy is in use, but it does not say which one. It is assumed that the applicable policy is clear from context. One would assume the policy itself explains in which context it applies. Finally, an explicit policy has a reference embedded in the signature to uniquely identify the version of the policy document in use. The reference consists of a unique ObjectIdentifier (URI or OID) and a hash computed over the document.

To compute the policy hash, the service needs a mapping from ObjectIdentifier to the policy file. This is done outside the service in a Signature Policy Provider. This type of provider can be configured on the same console page as the other security providers (like KeyStores). Many Signature Policy Providers can be created. The Signature Policy Provider parameter determines which one to use. The same provider can be shared by many XAdES Signature Verify Services, making it convenient to declare the mapping only once for multiple use.

An explicit policy is valid if the Signature Policy Provider can determine the path to the policy file and the hash of the contents of the file is equal to the hash that appears in the Signature. The hash guarantees the signer and verifier have the exact same document in their possession. Since the policy is not in machine readable form, no further checking is done.

The Signature Policy Provider should only map the policies that are deemed acceptable.

The policy document is not needed to verify an implied policy. Since the Signature Policy Provider is not involved, the service relies on the Accept Implied Policy boolean parameter to determine whether an implied policy is acceptable.

Finally, the Accept Unknown Properties boolean parameter indicates whether the verifier should accept unknown properties. This only affects the unsigned properties since the schema of signed properties is closed.

Edges:

The following table lists and describes the edges that are returned by the XAdES Digital Signature Verify service.

Edge	Description
success	The Signature is valid.
fail_parse	An iFL or XPath expression could not be evaluated.
fail_operation	The validation could not be performed.
fail_unsigned	The XML document is not signed.
fail_verify	The Signature is invalid.
fail_coverage	The application specified a node or attachment that should have been signed but was not covered by the Signature.

If the signature validates, the service will continue on the success edge. If validation fails because of Unsigned Attachments or incomplete Required Signature Coverage, the service will follow the fail_coverage edge. If validation fails for other reasons, the service will follow the fail_verify edge. If there is no signature, the flow will continue on the fail_unsigned edge.

Special Registers:

The following table lists and describes the special registers assigned upon successful validation of the signature.

Special Register	Description
xmldsig_signer	Holds the signer Distinguished Name
xmldsig_signer_cn	Holds the signer Common Name found in the signer Distinguished Name
xades_form	The XAdES Signature form. Possible values are XAdES-BES, XAdES-EPES, XAdES-T, XAdES-C.

Examples

For more information on related examples of XML Digital Signatures, see *XML Digital Signature Verify Service* on page 130. In particular, Example 3 on Certificate Revocation, Example 4 on Signature Coverage and Example 5 on Reducing Risks are directly applicable. The following examples cover topics specific to XAdES Signature verification.

Note: The examples in this section are specific to the XAdES Digital Signature Verify service (com.ibi.agents.XDXAdESVerifyAgent). For your convenience, the sample input and output documents are attached to the PDF in unabbreviated and unindented form.

For PDF-compatibility purposes, the file extension of the *XAdESVerify.zip* file is temporarily renamed to *.zap*. After saving this file to your file system, you must rename this extension back to *.zip* before the file can be used.

Example 1: Minimum XAdES Form

This example shows how to demand the presence of a signature policy by requesting the XAdES-EPES minimum form.

The following table lists the parameter values for the XDXAdESVerifyAgent. Other parameters that are not listed have their default value.

Parameter	Value
TrustStore Provider	trustprov
Minimum XAdES Form	XAdES-EPES

This assumes trustprov is the name of a KeyStore provider that contains the certificates of the Trusted CAs.

The XAdES-EPES form requires a signature policy. Since we did not configure a Signature Policy Provider, the service will only accept signatures with an implied policy. Refer to *Example 2: Explicit Signature Policy* on page 194 for an example with an explicit signature policy.

The first sample input document is shown as follows (indented for display purposes only).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Body>...</Body>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="" Id="xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556-ref0">
            <ds:Transforms>
               <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
            </ds:Transforms>
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>2HxElKLdnrWUim5oPiaRdHKlyJ2fQdDckdiaboWT7VI=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556-signedprops" Type="
http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>2GHN6Ndk5m25CcQLoLCbF/j9qiBTQHqoWxD9cFlab3c=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556-sigvalue">
i8OgEjg9dZpq2PsuDHrg9SpEizBvR/Q4iJH/e/xca1HOVo+ZMKz1Lu2nbCLh9A2YR0/TLRP3jMarrXz2QcMaMx2CxM6HhvFoMlpTifNMCQr
HY5iOhJ4hcfBCWlgymCXKdiKrYaYcsjjCWel/IJSqQ2V/8kALdXXmuEIP05AssWl=</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>
MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM...</ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="
#xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556" xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-055b92af-67ee-4c49-aefd-e6d7936a0556-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIlkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
               </xades:SignedSignatureProperties>
            </xades:SignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<eda>
   <error timestamp="2011-11-21T16:40:16Z" code="6" stage="AGENT" source="" removesig="false" trustprov="trustprov"
unsignedattachment="keep" xpathlevel="default" acceptunknown="false" maxpathlen="6" minform="XAdES-EPES" enablecrl="false"
keyusage="false" digestjceprov="NOT_SPECIFIED" acceptimplied="true">XAdES Signature validation failed: XAdES-BES is less than
the configured minimum XAdES form</error>
</eda>
```

This signature is in XAdES-BES form and will be rejected because it lacks a Signature Policy. An error document will be produced and the flow will continue on the fail_verify edge.

The second sample input document is shown as follows (indented for display purposes only).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Body Id="myid">...</Body>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="#myid" Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-ref0">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>e95XCAqcyE0xHBhZVSTL1zShbebjcvHB+7LVt14rOuo=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-signedprops" Type="
http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>krFeNaF03HOBTKeBVwXnBLjqUXyschFIFLOTpttwszY=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-sigvalue">
N2HU1hlMBX0ddBWMsyXdaqDbZFpePd6xGAYbPQ8nT9771ycneqhaSxQ26isKz/vYxbEqbG9CbE73RPQsWelt4vDuXvxYulmmwWxRa
OvnlhvL6mvFpVqyjFBlftyCxgGIKn59w9zzNJ4dLyxxD3nanNrEQaU3ENd1+PJEPql0T7Q=</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>
MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM... </ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="
#xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88" xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIlkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
                  <xades:SignaturePolicyIdentifier>
                     <xades:SignaturePolicyImplied/>
                  </xades:SignaturePolicyIdentifier>
               </xades:SignedSignatureProperties>
            </xades:SignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

This signature is in XAdES-EPES form and will be accepted. The output document is the same as the input document. The flow will continue on the success edge. The special register xmldsig_signer will be equal to CN=Partner SMIME Signature, O=Partner, C=US. The special register xmldsig_signer_cn will be equal to Partner SMIME Signature. Finally, the special register xades_form will be equal to XAdES-EPES.

Example 2: Explicit Signature Policy

This example shows how to require an explicit signature policy.

The following table lists the parameter values for the XDXAdESVerifyAgent. Other parameters that are not listed have their default value.

Parameter	Value
TrustStore Provider	trustprov
Minimum XAdES Form	XAdES-EPES
Signature Policy Provider	policyprov
Accept Implied Policy	false

This assumes trustprov is the name of a KeyStore provider that contains the certificates of the Trusted CAs.

The minimum XAdES form chosen requires a signature policy. Since the Accept Implied Policy parameter is false, this will require an explicit policy. The mapping between policy ObjectIdentifier (URI or OID) to the policy file must be configured in a Signature Policy Provider. Assume the provider policyprov has a mapping from the URI *http://iwaysoftware.com/xades#policy1.0* to the policy file policies/policy-1.0.doc. Assume that file contains the policy document.

The second input document in Example 1 will now be rejected because the policy is not explicit. A sample output of the service is shown as follows (indented for display purposes only):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<eda>
   <error timestamp="2011-11-21T16:40:16Z" code="6" stage="AGENT" source="" removesig="false" trustprov="trustprov"
unsignedattachment="keep" xpathlevel="default" acceptunknown="false" maxpathlen="6" minform="XAdES-EPES" enablecrl="false"
keyusage="false" digestjceprov="NOT_SPECIFIED" acceptimplied="true">XAdES Signature validation failed: XAdES-BES is less than
the configured minimum XAdES form</error>
</eda>
```

Another sample input document is shown as follows (indented for display purposes only).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Message>
   <Body Id="myid">...</Body>
   <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88">
      <ds:SignedInfo>
         <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2006/12/xml-c14n11"/>
         <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
         <ds:Reference URI="#myid" Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-ref0">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>e95XCAqcyE0xHBhZVSTL1zShbebjcvHB+7LVt14rOuo=</ds:DigestValue>
         </ds:Reference>
         <ds:Reference URI="#xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-signedprops" Type="
http://uri.etsi.org/01903#SignedProperties">
            <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
            <ds:DigestValue>krFeNaF03HOBTKeBVwXnBLjqUXyschFIFLOTpttwszY=</ds:DigestValue>
         </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-sigvalue">
N2HU1hlMBX0ddBWMsyXdaqDbZFpePd6xGAYbPQ8nT9771ycneqhaSxQ26isKz/vYxbEqbG9CbE73RPQsWelt4vDuXvxYulmmwWxRa
OvnlhvL6mvFpVqyjFBlftyCxgGIKn59w9zzNJ4dLyxxD3nanNrEQaU3ENd1+PJEPql0T7Q=</ds:SignatureValue>
      <ds:KeyInfo>
         <ds:X509Data>
            <ds:X509Certificate>
MIICejCCAeOgAwIBAgIBEDANBgkqhkiG9w0BAQUFADAxMQswCQYDVQQGEwJVUzENMAsGA1UEChM... </ds:X509Certificate>
         </ds:X509Data>
      </ds:KeyInfo>
      <ds:Object>
         <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="
#xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88" xmlns:xades141="http://uri.etsi.org/01903/v1.4.1#">
            <xades:SignedProperties Id="xmldsig-cb18fedb-ccf4-45ba-9ad6-15bcd6124b88-signedprops">
               <xades:SignedSignatureProperties>
                  <xades:SigningTime>2011-10-20T15:07:14.112-07:00</xades:SigningTime>
                  <xades:SigningCertificate>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>1uVIlkubDoJMZqOjmHLAUwVXKRD0WOJ8KRENz9SOwoM=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>16</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                     <xades:Cert>
                        <xades:CertDigest>
                           <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                           <ds:DigestValue>Q+wxGon0752PkZnMEBeNt976CrXZOXsA2v36ZhjPSvI=</ds:DigestValue>
                        </xades:CertDigest>
                        <xades:IssuerSerial>
                           <ds:X509IssuerName>CN=Example CA,O=iWay,C=US</ds:X509IssuerName>
                           <ds:X509SerialNumber>15</ds:X509SerialNumber>
                        </xades:IssuerSerial>
                     </xades:Cert>
                  </xades:SigningCertificate>
                  <xades:SignaturePolicyIdentifier>
                     <xades:SignaturePolicyImplied/>
                  </xades:SignaturePolicyIdentifier>
               </xades:SignedSignatureProperties>
            </xades:SignedProperties>
         </xades:QualifyingProperties>
      </ds:Object>
   </ds:Signature>
</Message>
```

This signature will be accepted if the policy-1.0.doc file contains the same policy document used by the signer. The output document will be the same as the input. The flow will continue on the success edge.

If the contents of the policy-1.0.doc file is modified, the hash will no longer agree and the signature will be rejected. The flow would continue on the fail_verify edge with an error document.

Authenticate/Impersonate Service

Syntax:

```
com.ibi.agents.XDPrincipalAgent
```

Description:

This service allows a process to authenticate and/or impersonate a user based on credentials in a message or message context.

Parameters:

The following table lists and describes the parameters for the Authenticate/Impersonate service.

Parameter Name	Description
Action	Determines what action this service performs. The following actions are available: ❑ **Authenticate.** Authenticates this user against the configured authentication realm. ❑ **Impersonate.** Push the principal, if authorized, to the channel (impersonate). ❑ **Pop.** Pop back one principal (cease impersonation). The limit is the initial principal.
Realm	The security realm to be used to validate this user.
User	User identity to be validated.

Parameter Name	Description
Credential	Credential (password) used to authenticate this user.

Edges:

The following table lists and describes the edges that are returned by the Authenticate/Impersonate service.

Edge	Description
success	The operation was successful.
fail_security	The credentials could not be authenticated when the selected action is set to *Authenticate* or *Impersonate*.
fail_parse	An iFL or XPath expression could not be evaluated.

Appendix A

Security Tools

This section provides information regarding security tools.

In this appendix:

Security Tools Overview

A common requirement is the development of a *secured* application. During the application development process, artifacts such as configuration files and process flows are produced. These artifacts must be protected to avoid tampering at the installation site.

Distributing the application requires that the flows, dictionary, and configuration files be signed, and that the policies distributed to the customer require that these file signatures be verified on use. This can be done by specifying the appropriate policies and then distributing the security file (in the configuration directory) with the application. If the distributed application does not grant administrative authority to any users, then the policy cannot be changed.

Management of the dictionary signature is automatic. A validly signed dictionary must be distributed to customers. Doing this simply means taking the dictionary to be run from the development system. No further preparation or action is necessary.

Process flows need to be signed individually before they are packaged for distribution.

The server manages signing keys. It considers two types of files, dictionaries and process flows. Each type uses a unique key pair. The server automatically selects the proper key for signing and validating configuration files based upon the type of file.

Signing Files

The Security Developers Tools extension (*iwdevkit*) includes a utility program for signing and verifying files. To run this utility, use the following server tool command:

```
>tool SignTree [-s|-v] <input file> [<output file>]
```

If the output file is omitted, the input file is rewritten with an XML signature. Use –s to sign (this is the default) and –v to verify.

Files that require manipulation can be located anywhere on the file system. For a given configuration, the dictionary is:

```
<iwayhome>\config\<configname>\<configname>.xml
```

As a convenience, the .xml suffix will be added by the utility if it is omitted. iWay home is the base of the iWay server installation.

Process flows are carried in the configuration processes subdirectory. They are named <name>_compiled_date.xml, and it is these that must be signed. The _gui and _image files are used for design time only, and are not needed at runtime.

The SignTree utility is also available as an Operating System command under the bin directory of the installation directory. On Windows, the batch file is called signtree.cmd. On Unix, the script is called signtree.sh. The command line options are the same as described in this section for the SignTree utility. This file should be placed into the iWay home directory. The command will only operate if the *iwdevkit* extension is in the extensions area. Naturally, the *iwdevkit* extension should not be distributed to customers.

Keeping Values Secret

All passwords that are required by iWay Service Manager (iSM), such as FTP login passwords, are stored within configuration files in masked format. Sometimes it is required to store application values in a secure manner. iSM tools provide the ability to store and hold property values using Advanced Encryption Standard (AES). The value is encoded using base64 as the encryption produces a binary encrypted value. Using a combination of techniques, no key or value ever needs to appear in clear form.

AES is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001. AES has been adopted by the U.S. government and is now used worldwide for non-secure and secure data. It supersedes the Data Encryption Standard (DES), which was published in 1977. The algorithm described by AES is a symmetric-key algorithm, meaning the same key is used to encrypt and decrypt the data.

The AES encryption used for storing and retrieving property values uses a 128-bit key. Methods exist to use longer keys and more complex operations, but for general purposes AES128 provides a strong and reliable encryption operation. It is not required that the iSM *set property* command be used to store the value. This command is provided as a convenience for users but any tool that stores an AES value can be used.

The iSM *set property* command is a common means of adding an encrypted value to a property file. Later, the _aes() iFL function can be used to retrieve this value for use in a process flow. The *set property* command is described in the *iWay Service Manager User's Guide*, while the use of _aes() is described in the *iWay Functional Language Reference Guide*.

The *-aes* keyword uses the provided key to encrypt the value. Advanced Encryption Standard (AES) is a strong encryption standard. The key must be 16 characters or less, as it becomes a 128-bit key.

For example:

```
set property testproperty userpswd mypassword -aes iwaykey1
```

If the key value is stored in a Special Register (SREG), which is the usual practice, named for the *secretkey* sample, the command would be structured as follows:

```
set property testproperty userpswd mypassword -aes _sreg(secretkey)
```

Either method would generate a properties file called *testproperty.properties*. A common practice is to load the key into a server-level configuration register. That register might itself be loaded from a property file or database encrypted with the *-encrypt* keyword that uses a built-in masking facility. Thus, the actual key never needs to appear in clear (legible) format. The key should, or course, not be stored in the same property file as the AES-encrypted values. For example, the following key line could be loaded into the *secretkey* SREG during iSM startup by its own _property() function:

```
mysecret=ENCR(3237310127231613138296)
```

The AES-encoded value might appear as follows:

```
#Saved by set property command
#Fri Jan 24 14:52:29 EST 2014
userpswd=A8vRBNezksAtoySgaFbOygkuMeYqmIy6v9GsIwU6K60\=
```

This value can be read using standard iFL functions in the specific operand or command. For example:

```
_aes('decrypt',_sreg(secretkey),_property('testproperty','userpswd'))
```

This example generates *mypassword* for use in the process execution.

iWay Service Manager (iSM) automatically handles base64 encoding and decoding as required.

Appendix B

nCipher Configuration

nCipher is a third-party provider of enterprise level hardware and software data protection and security solutions.

This section describes the nCipher configuration. The information is included solely to assist customers that are using their equipment.

Note: iWay does not endorse this equipment. Any questions concerning this equipment should be directed to the supplier.

In this appendix:

Provider Initialization (Validating Signatures)

By default, the SunPKCS11 provider only specifies mandatory PKCS#11 attributes when creating objects. The PKCS#11 library you are using will assign implementation specific default values to the other attributes. For example, when the SunPKCS11 provider imports a public key in the security token to validate a signature, the CKA_VERIFY attribute is absent and the nCipher default is false.

When this public key is passed to the C_VerifyInit() function, the public key is rejected with the error CKR_KEY_FUNCTION_NOT_PERMITTED as shown in the nCipher log. In Java, this can cause the following obscure error message:

```
ProviderException: initialization failed
```

The Sun PKCS11 Provider reference guide explains how to set specific attributes for various kinds of keys and cryptography operations. The simplest solution is to add the following line to the configuration file:

```
attributes = compatibility
```

The following is a relevant paragraph from the Sun PKCS#11 reference guide:

There is also a special form of the attributes option. You can write attributes = compatibility in the configuration file. That is a shortcut for a whole set of attribute statements. They are designed to provide maximum compatibility with existing Java applications, which may expect, for example, all key components to be accessible and secret keys to be useable for both encryption and decryption.

A sample configuration file using the attributes statement is shown in the following example:

```
name=nShield
library=D:/nfast/bin/cknfast.dll
description=nShield_PCI_500
slotListIndex=1
attributes = compatibility
```

SlotListIndex 0 is the HSM accelerator slot where MODULE protected keys are stored. A module protected key requires no password or user input. To use a softcard to protect your keys, you must define slotListIndex=1.

Java Configuration

On all machines with an nCipher card you must define a system environment variable, for example, CKNFAST_LOADSHARING=1. This allows the use of softcards to protect keys. After you have added the new variable you must reboot. For more information, refer to the nCipher documentation.

To configure SunPKCS11 insert the following line into the java.security file:

```
security.provider.3=sun.security.pkcs11.SunPKCS11 /nfast/sunpkcs11.cfg
```

Move all other providers up the numeric ordering list.

```
#
# List of providers and their preference orders (see above):
#
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=sun.security.pkcs11.SunPKCS11 /nfast/sunpkcs11.cfg
security.provider.4=org.bouncycastle.jce.provider.BouncyCastleProvider
security.provider.5=com.sun.net.ssl.internal.ssl.Provider
security.provider.6=com.sun.crypto.provider.SunJCE
security.provider.7=sun.security.jgss.SunProvider
security.provider.8=com.sun.security.sasl.Provider
```

Make sure the number you choose for SunPKCS11 is lower than bouncy castle and SunJCE.

Note: The PKCS11 driver is used to interact with the nCipher hardware. However, there is a Java limitation that effects the interaction. This limitation allows only a single PKCS11 driver to be defined. As a result, only a single instance of the PKCS11-based provider is supported in any given configuration.

Softcard for nCipher

FIPS 140-2 level 3 requires user authentication to access functions that use keys in the HSM. nCipher supports this by requiring you to present an operator smart card each time you ran an application that wanted to access some application keys. This may seem secure, but it is not the case when you are using an application, which runs continuously and without any intervention. Recently nCipher addressed this issue with the concept of a softcard. A softcard appears just like an operator smart card but is instead a file on the hard disk. A PIN is also associated with this softcard just like a normal smart card, but the advantage is that an application can run unattended.

Creating a Softcard

Assuming you have nCipher card installed as well as the software, you can run KeySafe to create a softcard:

Select *Start*, *All Programs*, *KeySafe*. On the left pane you will see a menu option for softcards. Select this option and then create a softcard.

You will be prompted for a name plus a PIN. You can list the slots using \nfast\bin\ckinfo to make sure you have your softcard installed as slot 1. You can list your softcards and you can run ckinfo from within \nfast\bin to show the slot configuration.

Multiple Softcards

If you create more than one softcard, it appears that the slot numbers are allocated incrementally from 1 and higher, based on the alphabetic listing of the softcard names. This will not be a problem if you have only one softcard. However, if you plan to use multiple softcards, you must make sure that you have the correct slotListIndex configured in the nCipher configuration file for the SUN PKCS11.

Key Creation Using Keytool

You can use the following command to create an RSA key using keytool:

```
keytool -storetype PKCS11 -keystore NONE -storepass clientcard -genkey
-keyalg RSA -alias myKey
```

where:

`clientcard`

Is a password for the softcard.

`myKey`

Is a private key alias that we are creating.

After you execute this command, you are prompted to provide your information to create the private key.

You can use the following command to view your keystore:

```
keytool -v -list -keystore NONE -storepass clientcard -storetype PKCS11
```

The following shows the sample output:

```
Keystore type: PKCS11
Keystore provider: SunPKCS11-nShield
Your keystore contains 1 entry
Alias name: maria
Entry type: keyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Maria, OU=iway, O=ibi, L=ny, ST=ny, C=US
Issuer: CN=Maria, OU=iway, O=ibi, L=ny, ST=ny, C=US
Serial number: 45face41
Valid from: Fri Mar 16 13:05:05 EDT 2007 until: Thu Jun 14 13:05:05 EDT
2007
Certificate fingerprints:
MD5: B3:1B:56:79:4A:45:AD:44:B2:F3:47:AF:ED:48:03:E4
SHA1: F8:BE:CD:85:A1:83:A9:10:FD:51:30:39:00:E0:83:9D:9B:4A:29:1C
```

Cryptography Provider

When you use an nCipher based keystore in your NAS2 configuration, you must select *SunPKCS11-nShield* as your cryptography provider. Also, it is recommended to select *BC* as your *S/MIME PKIX JCE Provider*.

Troubleshooting (PKCS11 RSA Private Key Exception)

Problem:

When using a PKCS11 device as a keystore within an NAS2 Sender configuration to sign a message, the following error message is displayed:

```
DEBUG (W.SmartSend.1) {com.ibi.agents.XDNAS2EmitAgent} Error emitting
NAS2: XD[FAIL] cause: 0 subcause: 0 message: java.io.IOException:
java.security.InvalidKeyException: Supplied key (sun.security.
```

```
pkcs11.P11Key$P11PrivateKey) is not a RSAPrivateKey instance

   at
org.bouncycastle.mail.smime.SMIMESignedGenerator$ContentSigner.write(Unkn
own Source)

   at
org.bouncycastle.mail.smime.handlers.PKCS7ContentHandler.writeTo(Unknown
Source)

   at
javax.activation.ObjectDataContentHandler.writeTo(DataHandler.java:883)
```

Solution:

The problem usually results from an incorrect configuration on the NAS2 Sender side for the S/MIME JCE Cryptography Provider. Confirm that you have selected a correct provider that corresponds to your device and are not using the default "BC" provider as your selection. The following are some examples:

Device: G&D Smart Card USB Key

Provider: SunPKCS11-StarSign

Device: nCipher HSM module

Provider: SunPKCS11-nShield

Note that each PKCS11 Device will have its own corresponding S/MIME JCE Cryptography Provider. However, all of the PKCS11 based provider names start with the SunPKCS11.

Appendix C

Authenticating an HTTP Client Using Kerberos

This section describes how to authenticate an HTTP client using Kerberos.

In this appendix:

Kerberos Overview

Kerberos is an authentication protocol that allows nodes communicating over a non-secure network to verify their identity in a secure manner. An HTTP client provider can be configured for Kerberos and later referred by name in an HTTP Emit service (XDNHttpEmitAgent) or an AS2 Nonblocking Emit service (XDNAS2EmitAgent).

When a server requires HTTP authentication, it returns one or more WWW-Authenticate headers listing the authentication schemes it can accept. The Authentication Preference parameter of the HTTP client provider defines which schemes take precedence over others. The value is a comma-separated list of authentication scheme names with the most preferred scheme listed first. The default value is:

```
negotiate,NTLM,Digest,Basic
```

where:

`negotiate`

Means SPNEGO and the only SPNEGO scheme implemented is Kerberos.

To disable all of the schemes except Kerberos, set the Authentication Preference parameter to *negotiate*.

The Kerberos Login Entry parameter of the HTTP client provider specifies the application logon entry in the JAAS configuration file that will be used to logon to Kerberos. This logon entry configures a Kerberos logon module (Krb5LoginModule). The Krb5LoginModule contains numerous options that can be configured in the JAAS configuration file. For more information about the available options for the Krb5LoginModule, see the Javadoc.

Kerberos Authentication

Kerberos authentication requires the following system properties to be set:

- `java.security.krb5.conf`

  The path to the Kerberos configuration file (usually named `krb5.conf`).

- `java.security.auth.login.config`

  The path to the JAAS configuration file (usually named `login.conf`).

The following system property can also be set to *true* to enable Kerberos debugging:

```
sun.security.krb5.debug
```

Sample Kerberos Configuration File

The following is a sample Kerberos configuration file:

```
[libdefaults]
    default_realm = IBI.COM
    udp_preference_limit = 1
[realms]
    IBI.COM = {
        kdc = ADSERVER.IBI.COM
    }
[domain_realms]
.ibi.com=IBI.COM
ibi.com=IBI.COM
```

Sample JAAS Configuration File

The following is a sample JAAS configuration file:

```
iWayHttpClient {
  com.sun.security.auth.module.Krb5LoginModule
     required
     // debug=true
     useKeyTab=true
     storeKey=true
     doNotPrompt=false;
};
```

The debug option of the Krb5LoginModule can be used in conjunction with the `sun.security.krb5.debug` system property to offer maximum debugging information. In this sample file, the debug option is commented out.

The `doNotPrompt` property is set to *false* to allow the HTTP client provider to provide the user name and password through a JAAS Callback. The user will not be prompted.

Kerberos Troubleshooting

This section provides several troubleshooting tips for Kerberos.

- Ensure that you have properly configured the krb5.conf file.

  The krb5.conf file is used to describe the Kerberos realm to be used for authentication and the location of the Key Distribution Center (KDC). This file has the following structure:

  ```
  [libdefaults]
     default_realm = MYCOMPANY.COM
     udp_preference_limit = 1
  [realms]
     MYCOMPANY.COM = {
      kdc = MYREALM.MYCOMPANY.COM
  }
  [domain_realms]
  .MYCOMPANY.com=MYCOMPANY.COM
  MYCOMPANY.com=MYCOMPANY.COM
  ```

  In this example, the Kerberos realm is MYCOMPANY.com and the KDC is located at MYREALM.MYCOMPANY.COM. Additional mapping information is provided in the [domain_realms} section.

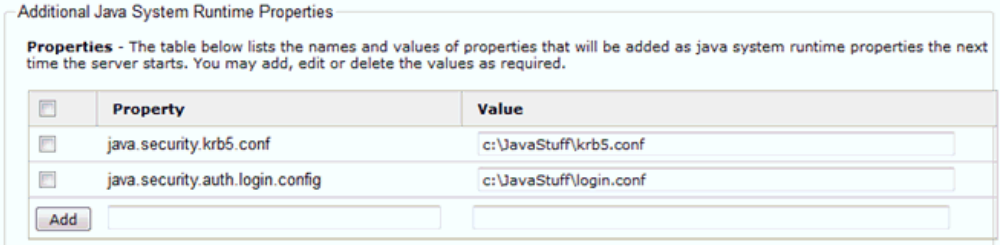
- Ensure that you have a properly configured the login.conf file.

  The login.conf file is used to configure the authentication mechanism used by Java Authentication and Authorization Service (JAAS). This file has the following structure:

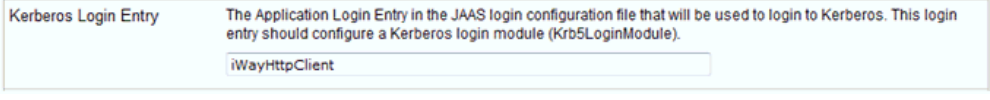
  ```
  iWayHttpClient { com.sun.security.auth.module.Krb5LoginModule
    required
    // debug=true
    useKeyTab=true
    storeKey=true
    doNotPrompt=false;
  };
  ```

  In this example, iWayHttpClient is the name to be used by all iWay applications (for example, iSM). The com.sun.security.auth.module.Krb5LoginMobile entry instructs iSM to use the Kerberos 5 login module. If you want to debug the Kerberos authentication process, then uncomment the debug=true statement.

- Ensure that you specify the location of the krb5.conf and login.conf files for your instance of iSM. You must provide the location of the krb5.conf and login.conf files in the Additional Java System Runtime Properties section of the iSM console.



- Ensure to specify which entry to use in the login.conf file. You must specify which entry to use for the HTTP Pooling Provider in login.conf, even if there is only one entry present. This is configured in the HTTP Pooling Provider properties section of the iSM console.



Resolving the Unable to Load Configuration File Error

You may encounter a "Could not load configuration file c:\Windows\krb5.ini (the system cannot find the file specified)" error message. For example:

```
[2011-11-16T12:37:40.998Z] ERROR (W.Retrieve_CRMChannel.1)
W.Retrieve_CRMChannel.1: [RequestTargetAuthentication - process()] -
Authentication error: Invalid name provided (Mechanism level: Could not
load configuration file C:\Windows\krb5.ini (The system cannot find the
file specified))
```

The following workarounds are available to resolve this error:

1. Ensure that you properly configured the krb5.conf and login.conf files as described in this appendix. Also, ensure that you have properly specified their location in the Additional Java System Runtime Properties and pointed to the correct login.conf entry in the HTTP Pooling Provider properties.

2. If this error is still generated after you have checked the properties described in step 1, then you must copy the krb5.conf and login.conf files to the <Java Home>\libs\security directory (on Windows) or <JavaHome>\etc (on Linux). Java looks for the krb5.conf and login.conf files by first checking if the location of these files has been explicitly listed. If for some reason (such as file permissions) Java cannot read the files in the location specified, Java then looks in the <Java Home>\libs\security directory. If the files are not found in this location, Java then defaults to c:\Windows\krb5.ini (for Windows), which results in the error message described above.

Appendix D

Configuring Kerberos With Microsoft SQL Server

This section describes how to configure Kerberos with Microsoft SQL (MS SQL) Server.

In this appendix:

Hardening the Java Virtual Machine Cryptography

Encrypting Kerberos requires you to replace the default encryption .jar files in your Java Virtual Machine (JVM) with the unlimited strength editions. To harden the JVM cryptography:

1. Obtain a copy of the following archive:

   Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8.

2. Unzip this archive and extract the following .jar files to the location of the JVM runtime folder:

   - local_policy.jar
   - US_export_policy.jar

   Note: You can move or replace the current jar files in the folder.

   On Windows, the JVM runtime folder location is:

   ```
   {jre folder}/lib/security
   ```

   On Linux, the JVM runtime folder location is:

   ```
   {jre}\lib\security
   ```

3. Ensure that the correct runtime JVM folder is copied, as there may be multiple versions of java on a machine.

   If the jar files are not correct, the following error will appear when connecting to Kerberos:

   ```
   Integrated Authentication Failed{guid number}
   ```

Using the Java Authentication and Authorization Service

The Java Authentication and Authorization Service (JAAS) is one of several Java plug in modules for security. This allows applications to use security services without having to hard-code values in the source code of a program. One of the plug in modules is for Kerberos. A subject is a set of credentials representing a single entity. For a given subject, a principal, representing the specifics of the type of subject, such as KRB_NT_PRINCIPAL, describes the formatting and representation of the credentials. A subject when passed to the Login module goes through the following states:

- Initialization (any error is initialized)
- Login
- Commit for success, or abort with logout (the final stage for both)

There are multiple checks during Kerberos initialization. The Subject is usually the complete FQDN (Fully Qualified Domain Name) of the user ID and the Kerberos TGT ticket attached to it. The Principal is the FQDN of the user ID in user principal name format: domainuser@REAL.COM

The JAAS service uses different modules and parameters for *pluggable* security. The Kerberos module is called K5b5LoginModule and has the property *required* (there are properties for usage such as *optional* and so on). The JAAS is used here solely for authentication purposes.

Creating a JAAS File for the SQL Server Driver for Kerberos

In this sample configuration file, the SQL Server Driver is being configured to use the Krb5LoginModule, requiring its use and failing to start the driver if the login module fails. The debug mode is currently set to:

```
ON "debug=True"
```

The login modules are called in a callback. Using the following syntax ensures that no screen texts appear asking for credentials, and fails if the credentials are not found:

```
doNotPrompt=true
```

The following Kerberos default ticket cache will not be used.

```
useTicketCache=false
```

The Kerberos authentication key should be imported and stored in memory, while the credentials come from the keyTab. Optionally, the user principal can be typed in the JAAS file as *user@REALM1.REALM2.COM*, overriding the keyTab. The following term is required:

```
SQLJDBCDriver
```

On Linux, the term must be in lowercase letters.

You can save iwjaas.conf on either Windows or Linux.

```
SQLJDBCDriver
{
com.sun.security.auth.module.Krb5LoginModule required
//  refreshKrb5Config=false
debug=true
doNotPrompt=true
useTicketCache=false
useKeyTab=true
keyTab="sqls6.keytab"
//      principal="user@REALM.COM";
};
```

Note: On Linux, *sqljdbc* must by in lowercase letters.

Each login module configuration file entry consists of a name followed by one or more LoginModule-specific entries, where each LoginModule-specific entry is terminated by a semicolon, and the entire group of LoginModule-specific entries is enclosed in braces. Each configuration file entry is terminated by a semicolon.

Uncomment refreshKrb5Config to start off with a clean configuration only when debugging.

The following table lists and describes the entries.

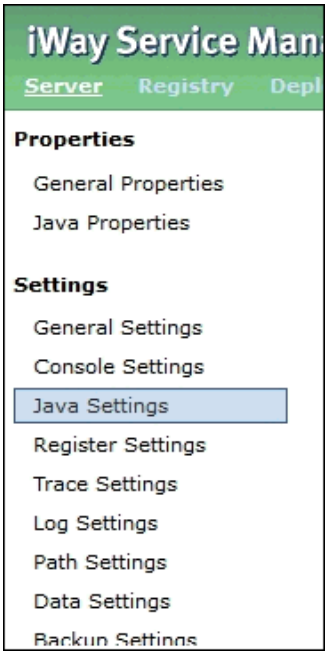
Entry	Description
debug=true	Turns on debug mode.
doNotPrompt=true	Do not prompt user for credentials.
useTicketCache=false	Do not use the Kerberos default cache.
useKeyTab=true	Use a keytab instead of the cache.
keyTab	Keytab name. Paths in this name do not always work.
principal	Override the principal in the keytab with this value debug only. **Note:** All names in the principal name *must* exist.

Configuring iWay Service Manager Run Time for Kerberos

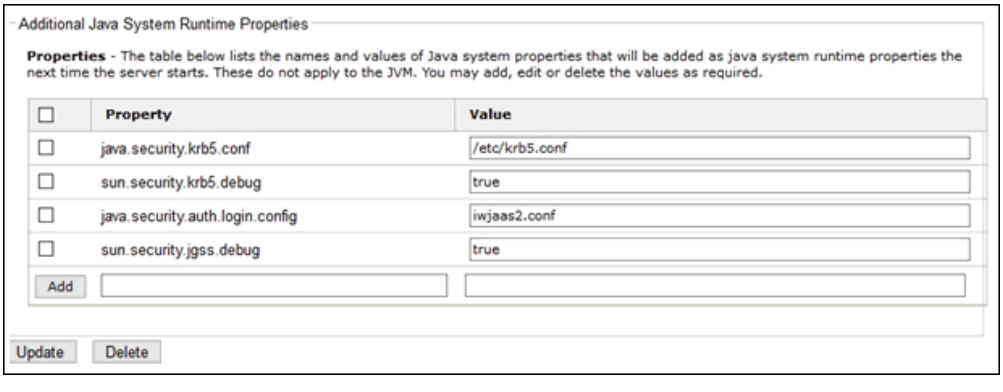
This section describes how to configure iWay Service Manager (iSM) runtime for Kerberos.

Procedure: How to Configure iSM Runtime for Kerberos

1. Start iSM using he appropriate command file on Windows or Linux.
2. Using a web browser, access the iSM Administration Console.
3. Click *Java Settings* in the left pane, as shown in the following image.



4. In the Java Settings page that appears, under the Additional Java System Runtime Properties section, enter and/or select the property names and the appropriate values.



The following table lists and describes the required properties.

Property	Description
java.security.krb5.conf	The path and file name of the Kerberos configuration file.
sun.security.krb5.debug	Enables Kerberos debugging, disable after initial testing.
java.security.auth.login.config	The JAAS configuration file, for example, iwjaas.conf.
sun.security.jgss.debug	Debugging for the JAAS.

5. If you wish to add a property and value, enter the property and value next to the Add button, and then click *Add* after each entry combination.
6. When you are complete, click *Update*.
7. When you are finished, return to the iSM Administration Console, completely stop and then restart iSM.

Appendix E

Configuring Microsoft SQL Server JDBC Driver Version 6 with Kerberos

This section describes how to configure Microsoft SQL (MS SQL) Server JDBC Driver version 6 with Kerberos on Windows and Linux.

In this appendix:

Prerequisites for Windows Active Directory

Before configuring Microsoft SQL Server JDBC Driver Version 6 with Kerberos, ensure that the following prerequisites are configured on your environment.

Servers:

- Active Directory Domain Services
- Windows Domain on Windows Server 2008
- Windows Server 2012
- Kerberos authentication enabled Windows Server 2012R2 (one Primary Domain Controller, and at least one Secondary Controller)

Clients Joined to the Domain:

- SQL Server servers running Windows 2012R2
- client workstations

Configuring Microsoft SQL Server JDBC Driver Version 6 With Kerberos Using Windows 2008/Windows 2012

To configure Microsoft SQL Server JDBC Driver Version 6 with Kerberos, using Windows 2008 or Windows 2012, in the Domain Controller, open *Active Directory Domains and Trusts*, then right-click the active domain and select *raise domain functional level*.

This enables the domain to support delegation and SPNs (Service Principal Names), as well as higher encryption levels.

If there is more than one domain controller, you will have to wait until the changes are propagated to the other domain controllers. You can confirm this by examining the Windows Event Log on the primary domain controller, and opening the Windows Event log. Scroll down and expand *Applications and Services Logs*, select *DFS Replication* and ensure no error messages have occurred. Additionally, check the information messages for a time stamp near your changes. Look for Event ID 1206, the text should have the name of the PDC (Primary Domain Controller) and a success message.

For more information, see the *Raise the Domain Functional Level* section in the Microsoft Developer Network website at *https://msdn.microsoft.com/en-us/library/cc753104(v=ws.11).aspx*, or see the Kerberos Enhancements section at *https://technet.microsoft.com/library/cc749438(ws.10).aspx*

Setting Up Accounts for the SQL Server

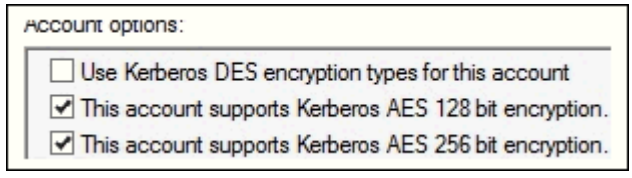
You can use Active Directory Users and Computers to create two domain user accounts for SQL Server in Active Directory: one for the SQL Server database account, and one (if needed) for the SQL Serve Reporting services.

Configuring User Account Attributes

Perform the following tasks for each of these accounts created for SQL and Kerberos:

- ❑ In the User *Account* property page, in the *Active Directory Users and Computers* section, ensure that the property check box for *Account is sensitive and cannot be delegated* is cleared.

- For the encryption types, make sure only the AES encryption types are selected, as shown in the following image.



Procedure: How to Configure the Active Directory Domain Controller for Delegation

1. On the primary Domain controller, log on and start the server manager.
2. From the Tools menu, select *Active Directory Users and Computers*, then right-click the computer you wish to set up for delegation (the SQL Server computer), and select *Trust this computer for delegation*.

   If the computer that is running SQL Server is the last computer contacted, but that computer has a linked server, it must also be granted delegation permissions. If it is not the last computer in the chain, all the computers that are intermediaries must be trusted for delegation.
3. Perform the following steps to grant delegation permission to the SQL Server service account domain user account. You must have a domain user account for clustered SQL Server installations (this step is not required for computers that are running SQL Server and using a local system account).
   a. In the Users folder, right-click the user account, and then click *Properties*.
   b. In the user account properties dialog box, click the Account tab.
   c. Under Account Options, select the *Account is Trusted for Delegation* check box, and ensure that the check box for *Account is sensitive and cannot be delegated*, is cleared for this account.

Configuring User Account Security Attributes

After installing and configuring SQL Server, you are returned to the Active Directory Domain Controller where you will need to configure the account properties of the MSSQL user.

Procedure: How to SQL Server on a Linux Machine

Procedure: How to Install SQL Server on Windows

1. Install the SQL Server (Enterprise, Developer, Evaluation, Business Intelligence, or Standard editions only) on a server in the same domain as the domain controller.

For more information, see the Microsoft support matrix for additional features at *https://msdn.microsoft.com/en-us/library/cc645993(v=sql.120).aspx#Enterprise_security*.

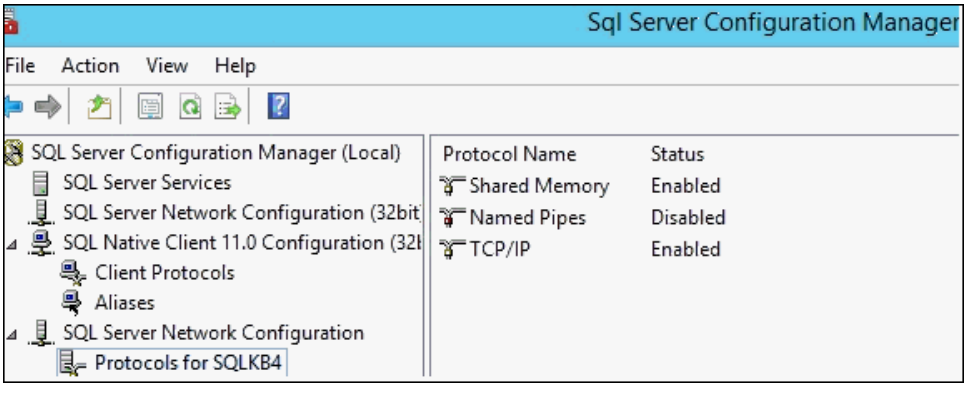
You can also see the Microsoft Quick Start Installation guide at *https://msdn.microsoft.com/en-us/library/bb500433(v=sql.120).aspx.*

2. Supply the domain accounts for the SQL Server service that you created in the previous steps on the domain controller, and for the SQL Server Analysis Services (if used).

   The SQL server must be installed on a Windows or Integrated authentication only. It is not possible to have a secure Kerberos connection mode when there can be a backdoor through a a native mode account. In *Integrated mode* the *SA* account is present but disabled.

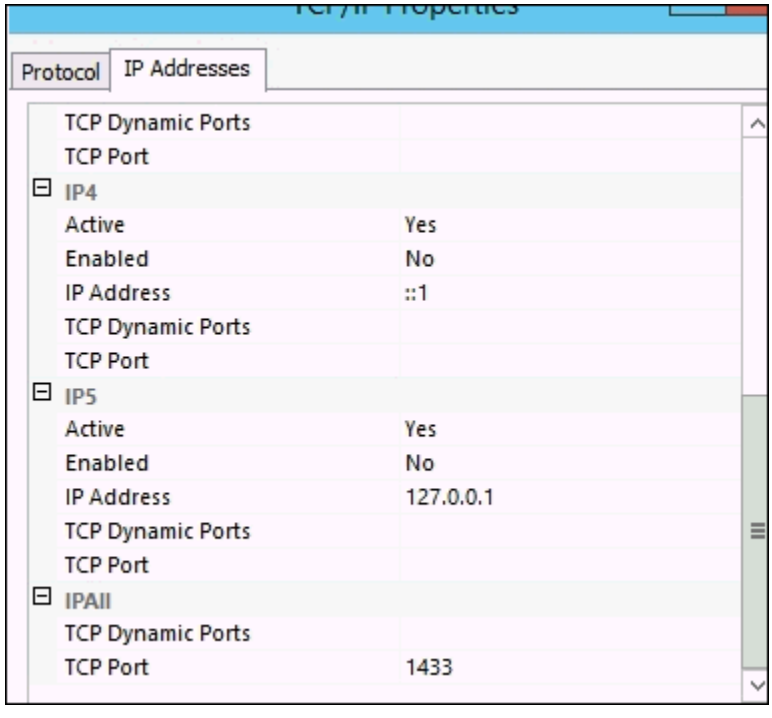
Procedure: How to Configure SQL Server After Installation

1. When the installation is complete, expand the *SQL Server Configuration Manager*, expand *SQL Server Network Configuration*, then select *TCP/IP* under Protocol Name on the right pane, as shown in the following image.



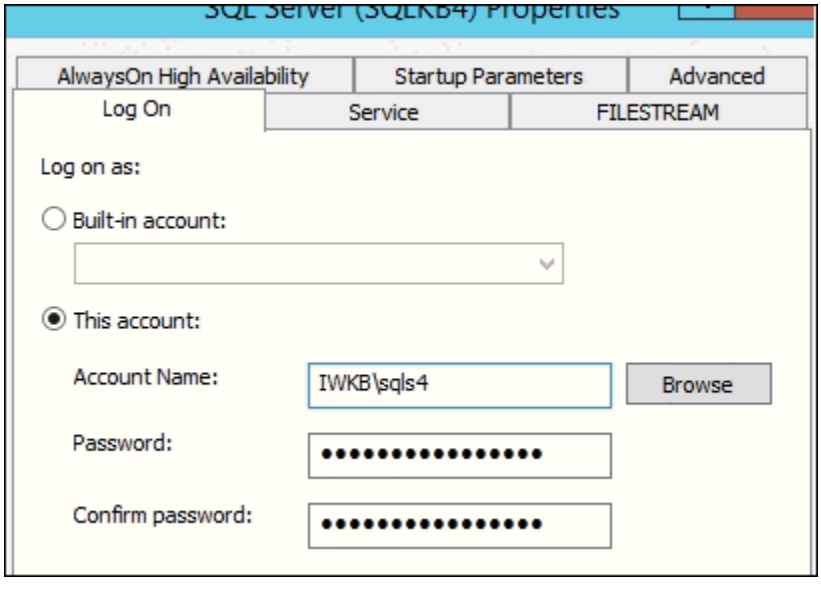
The TCP/IP Properties dialog opens.

2. Click the IP Addresses tab and ensure that the *TCP Dynamic Ports* property is blank for all entries, then enter *1433* in the TCP Port field of the IPALL section, as shown in the following image.



3. Click *OK*.

4. Click on the SQL Server Services in the same SQL Server Configuration Window and ensure that the domain account is the account selected for the SQL Server service, as shown in the following image.

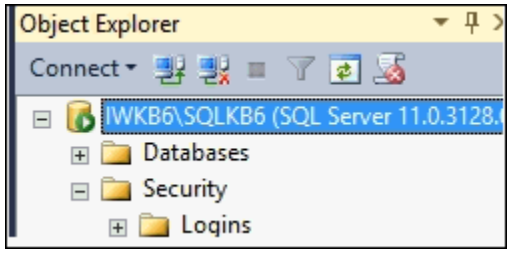


5. If a change is made in this property, click *Apply* then click *OK*, and restart the service to accept the change in the server.

Procedure: How to Add the SQL Server Domain Account to SQL Server Management Studio

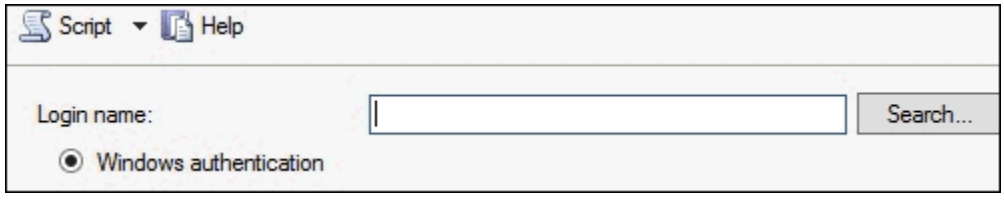
1. Log on as an Administrator to the server with SQL Server installed.
2. Open the *SQL Server Management Studio* application and select the current server when prompted for the database engine, as shown in the following image.

The Object Explorer opens, as shown in the following image.

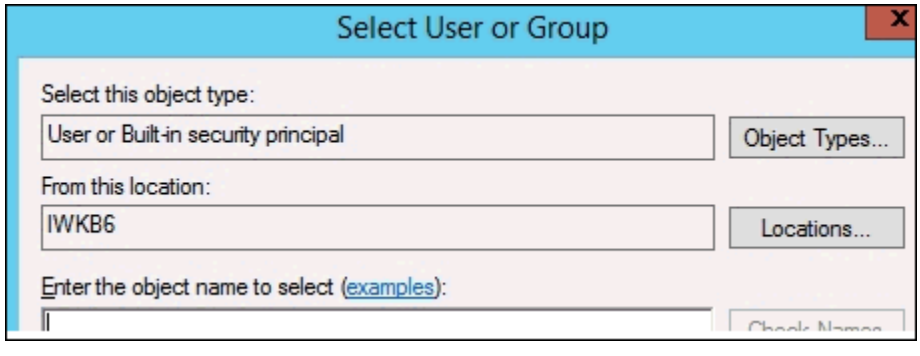


3. Expand the *Logins* folder.
4. Right-click on *Logins* and select *New Login*.

   The Login New dialog opens.
5. Select the *Windows Authentication* radio button, then click *Search*, as shown in the following image.

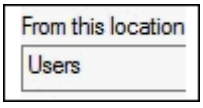


The *Select Window or Group* dialog opens, with the current machine name already indicated in the *From this location* field, as shown in the following image.



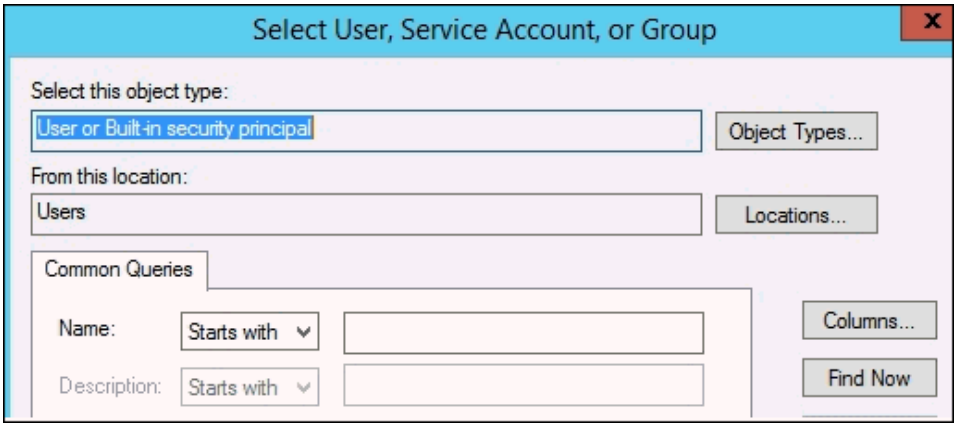
6. Click the *Locations* button, and expand it until the domain of the machine and all of the built-in objects of the domain appear.
7. Click *Users*, and then click *OK*.

The *Select User or Group* dialog returns, but the *From this location* field now shows *Users* instead of the local machine name, as shown in the following image.



8. Click the *Advanced* button at the lower left of the dialog.

   The *Select User, Service Account, or Group* dialog opens, as shown in the following image.



9. Click the *Find now* button.

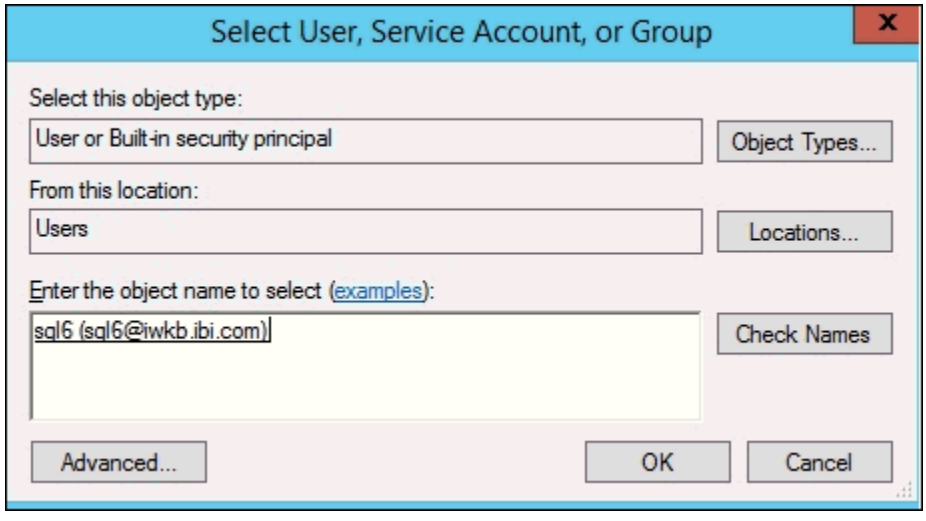
   The list box fills with the users of the domain.

10. Select the User ID that is running an instance of SQL Server on the current machine.

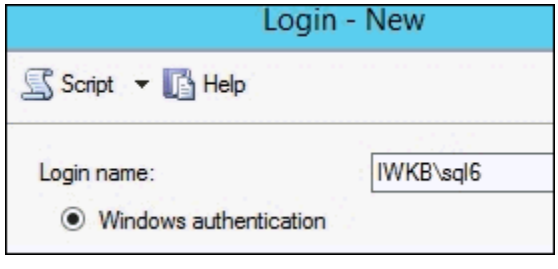
11. Click on the name of the SQL Server domain account name and click *OK*.

    The Find property page closes and the list box of the *Select User, Service Account, or Group* dialog now displays the selected account in user principal name format.

12. Click *OK* to close the dialog, as shown in the following image.



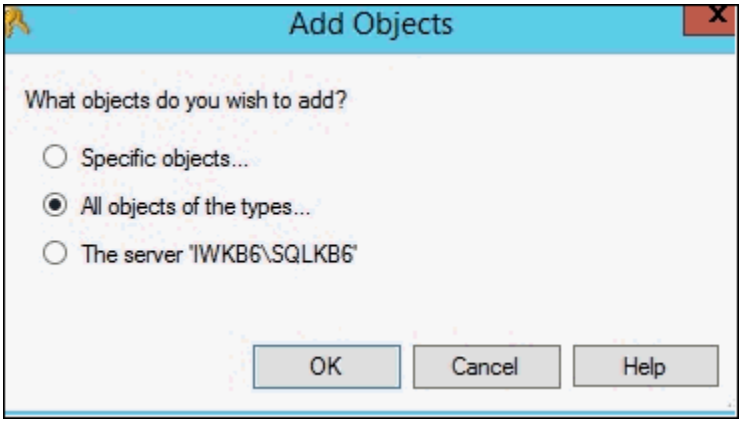
The Login New dialog now shows the SQL server user in Windows 2000 format Domain \User, as shown in the following image.



On the left side of the dialog, the *Select a page* section is positioned on General.

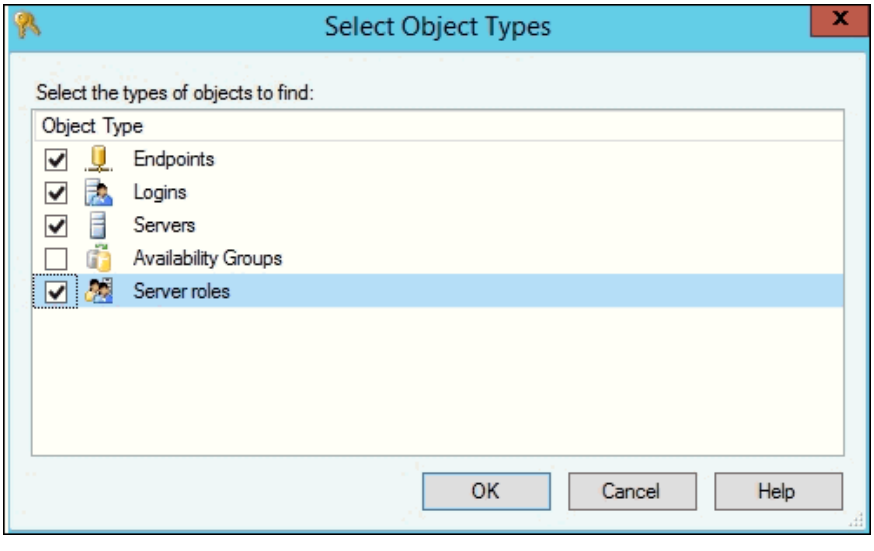
13. Click *User Mapping* and select the database or databases to be used with the ID on remote connection.
14. Use the *Select a page* to move to *Securables*.
15.
16. Click the *Search* button.

The Add Objects dialog appears, as shown in the following image.



17. Select *All objects of the types...* and then click *OK*.

    The Select Object types dialog opens, as shown in the following image.

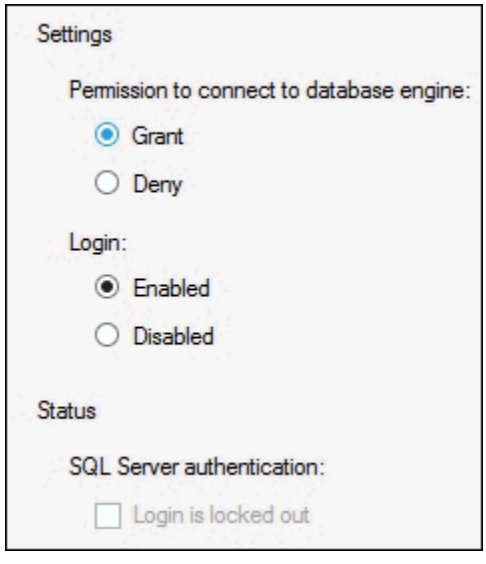


18. Select the following check boxes, and then click *OK*.

    - *Endpoints*
    - *Logins*
    - *Servers*

- *Server roles*

19. In the object selections, ensure that the Permissions for *Authenticate Server*, *Connect SQL*, *View database*, and any other relevant permissions are selected.
20. Use the Select a Page tool to move to *Status* and make sure that certain options in the Settings section, shown in the image below, are enabled.



This must be done for the User for SQL Server connection and may be done for additional regular SQL Server users.

Procedure: How to Create Service Principal Names

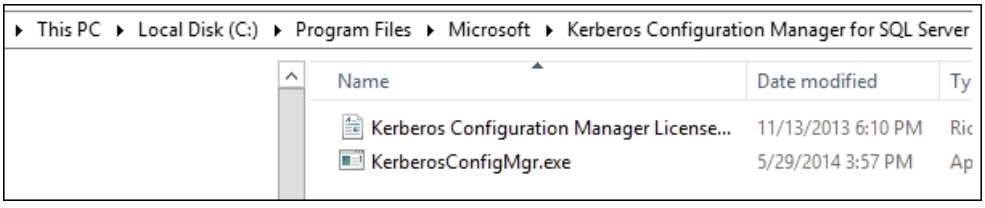
1. Using your web browser, search for and download the Microsoft Kerberos Configuration Manager for SQL Server, and then install it to your machine with administrator permissions.

   You can install it into the following drive:

   ```
   C:\Program Files\Microsoft\Kerberos Configuration Manager for SQL Server
   ```

   No shortcuts or entries are added to the windows Start menu.

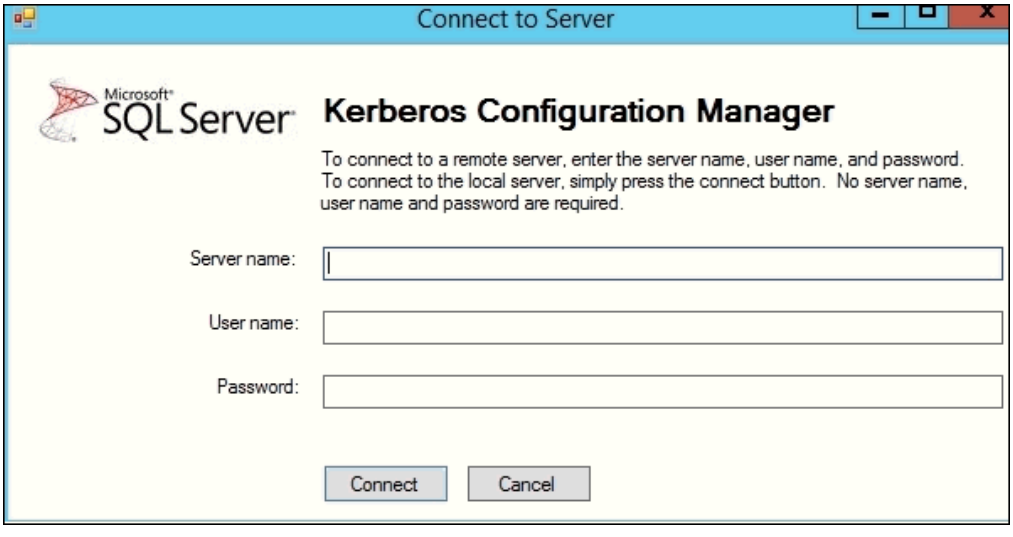
2. Navigate to the location using the windows File Explorer, right-click on the KerberosConfigMgr.exe, and select *Run as Administrator*.



The Configuration Manager window opens.

3. Click *Connect* from the menu bar.

A dialog window opens with the connection parameter credential fields. *Do not* enter anything into the fields, as shown in the following image.



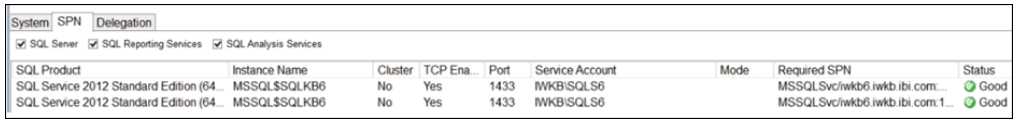
4. Click *Connect*.

The following tabs appear:

- System
- SPN
- Delegation

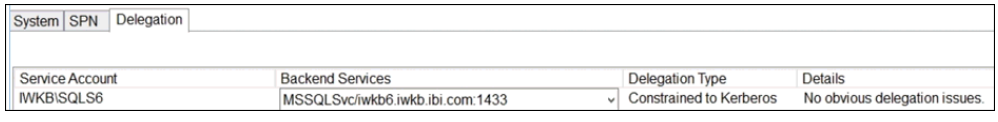
The System tab shows the connection information on the machine with SQL Server installed.

The SPN tab allows you to see the Instance Name and the associated Service Account. You can scroll to the far right to see the Required SPN. If the SPN is not present in the Active Directory, then the two buttons *Fix* and *Generate* will be enabled. The Generate option will dynamically create the SPNs on the system and link them to the account. You can perform this action for as many SPNs as are listed in red (if *Fix* is selected, the changes are written to a file to be executed at a later time). It can be useful to keep a record of the changes, since the will be generated anyway.



When finished, the Status column will have a green check box and the word *Good* for all SPNs and accounts.

5. Click *Exit* on the menu bar (the changes that were made can be saved as an XML document by selecting *Save* from the menu bar and providing a file name) to end the utility.



Registering Manual SPN

Enter the following syntax as a template on the Domain controller:

```
setspn -A MSSQLSvc/myhost.redmond.microsoft.com:1433 accountname
```

Your host and domain name should be specified. The account name is the domain user account, which uses the principal name format.

For named instances, use the following format:

```
setspn -A MSSQLSvc/myhost.redmond.microsoft.com:1433 accountname
```

Procedure: How to Configure Security to the SQL Server Service to Delegate Using SPN

1. From the main domain controller in Active Directory Users and Computers, right-click the computer that you wish to set up for delegation, and then click *Trust this computer for delegation*.

If the computer that is running SQL Server is the last computer contacted (and the computer has a linked server), the it must also be granted delegation permissions. If it is not the last computer in the chain, then all the computers that are intermediaries must be trusted for delegation.

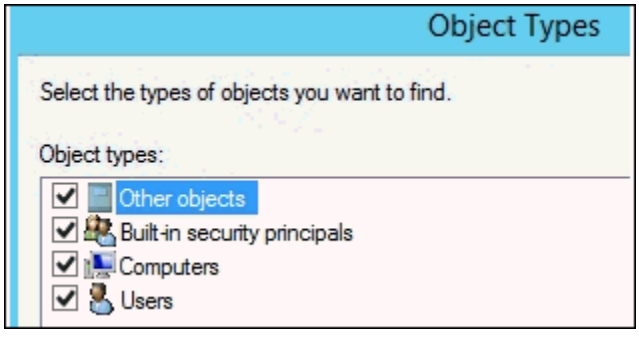
2. Grant delegation permission to the SQL Server service account domain user account.

   You must have a domain user account for clustered SQL Server installations.

   Note: This step is not required for computers that are running SQL Server and using a local system account.

3. In the Users folder, right-click the user account, and then click *Properties*.
4. In the user account properties dialog box, click the Account tab.
5. Under Account Options, select the *Account is Trusted for Delegation* check box and make sure that the *Account is sensitive and cannot be delegated* check box is cleared for this account.
6. In the Delegation tab, select the *Trust this user to specified Services only* check box, and then underneath, select *Use Kerberos only*.
7. In the *Services to which Account can present Delegation* list, click *Add*, and then click the *Users or Computers* button.
8. In the Select Computers or Users Dialog, click the *Object Types* button to show the list of object types to select.

   The following object types are selected by default.



9. In the *From this location* field, ensure that the current domain (and not the current machine) is selected.
10. Click *Advanced*.
11. In the Select Users or Computers dialog that appears, click *Find Now* to display a list of computers and users.

12. Select the user name that was used to run SQL Server, to map the Service Principal Name to the user object, and complete the Delegation process.

    You are returned to the Delegation page of the Account properties, with one or more entries in the service principal name list.

    You can select all or as many entries as you wish, and then click *Add* to register the selection in Active Directory.

Procedure: How to Configure the SQL Server Service to Create SPNs Dynamically

The JDBC driver needs these permissions. Do not perform step 12 unless dynamic SPNs are enabled throughout the enterprise.

To enable the enterprise, you must grant the following access control settings for the SQL Server service account in the Active Directory directory service:

- Read servicePrincipalName
- Write servicePrincipalName

Important:

- If you use the Active Directory Service Interfaces (ADSI) Edit snap-in, and you incorrectly modify the attributes of Active Directory objects, serious problems can occur. To resolve these problems, you may have to reinstall Microsoft Windows Server and Active Directory. It cannot be guaranteed that these problems can be resolved. Modify these attributes at your own risk.
- You must be logged on as a domain administrator. Alternatively, you must ask your domain administrator to grant the appropriate permissions and the appropriate user rights to the SQL Server startup account.

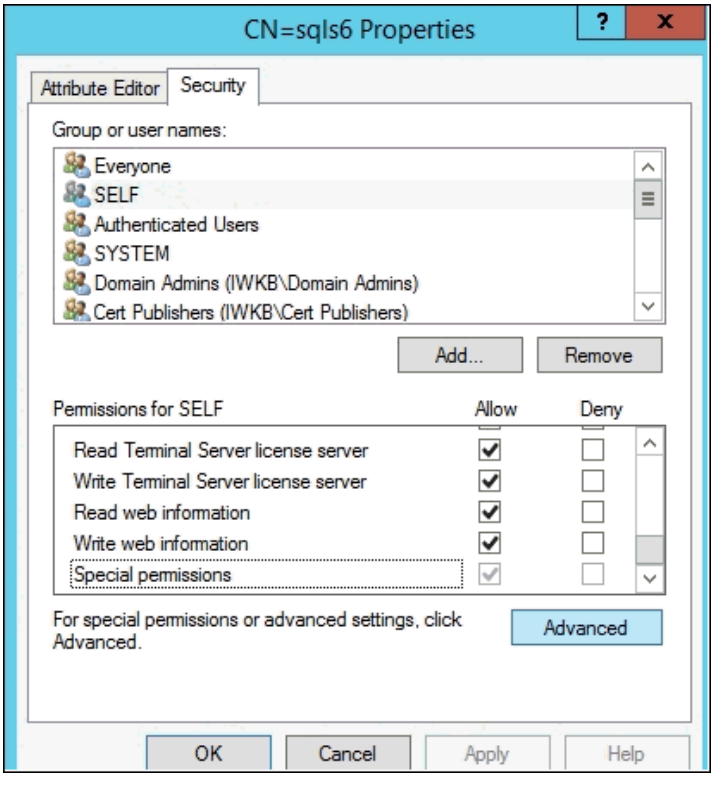
Perform the following steps to configure the SQL Server service to create SPNs dynamically when the SQL Server service starts:

1. Click *Start*, select *Run*, then type *Adsiedit.msc* in the field, and click *OK*.
2. In the ADSI Edit pane, expand *Domain [DomainName]*, expand *DC= RootDomainName*, expand *CN=Users*, right-click *CN= AccountName*, and then click *Properties*.

   Notes:

   - *DomainName* is a placeholder for the name of the domain.
   - *RootDomainName* is a placeholder for the name of the root domain.
   - *AccountName* is a placeholder for the account that you specify to start the SQL Server service.

- If you specify the Local System account to start the SQL Server service, then the *AccountName* is a placeholder for the account that you use to log on to Microsoft Windows.
- If you specify a domain user account to start the SQL Server service, then the *AccountName* is a placeholder for the domain user account.



3. In the CN= AccountName Properties dialog box, click the Security tab.
4. On the Security tab, click *Advanced*.
5. In the Advanced Security Settings dialog box, make sure that SELF is listed under Permission entries. If SELF is not listed, click *Add*, and then add *SELF*.
6. Under Permission entries, click *SELF*, and then click *Edit*.
7. In the Permission Entry dialog box, click the Properties tab.
8. On the Properties tab, select *This object only* in the *Apply onto* list, and then select the following check boxes under Permissions:
   - *Read servicePrincipalName*

- ❑ *Write servicePrincipalName*

Important: There are many properties and each must be searched for in the list. If they do not appear, the wrong object is selected or delegation is not enabled on the account.

9. Click *OK*.
10. In the next dialog that appears, click *OK* again.
11. In the CN= AccountName Properties dialog box, click *Attribute Editor*.
12. Under Attributes, click *servicePrincipalName* in the Attribute column, and then click *Edit*.
13. **This step is for Dynamic SPNs only:** In the Multi-valued String Editor dialog box, remove the service principle names (SPNs) for the instances of SQL Server that use this SQL Server service account.

    Important: You should only delete the SPNs for the instances of SQL Server that you are currently working on. The other instances of SQL Server that use this service account will be able to remove the SPNs that are related to these instances the next time that you start these instances.
14. Exit the ADSI Edit snap-in.

    Dynamic SPNs eliminate problems if you change the TCP/IP port or the domain name for new installations of SQL Server, or for existing instances of SQL Server (though they are harder to diagnose). For more information, see *SQL Server Clustered Server Warning* on page 248.

Connection Rule for NTLM and Kerberos

If a user logs on to SQL Server management studio with a domain account, and the SQL Server database is installed on the same machine, then the *NTLM* authentication mode is always used. If a user logs on with a domain account, and connects to a SQL Server database on another machine, then Kerberos is used instead of NTLM. JDBC access should always be on a separate machine from the SQL database and use Kerberos authentication.

For more information, see the Choosing an Authentication Mode section in the Microsoft Tech Net website at: *https://technet.microsoft.com/en-us/library/ms144284(v=sql.105).aspx*.

Using Kerberos Authentication With SQL Server

The following conditions apply when using Kerberos authentication with SQL Server:

- ❑ The client and server computers must be part of the same Windows domain, or in trusted domains.

- A Service Principal Name (SPN) must be registered with Active Directory, which assumes the role of the Key Distribution Center in a Windows domain. The SPN, after it is registered, maps to the Windows account that started the SQL Server instance service. If the SPN registration fails or has not been performed, then the Windows security layer cannot determine the account associated with the SPN, and Kerberos authentication will not be used.

Cross domain SQL Server usage and authentication does not apply in this use case. For more information, see the appendix for where to go next on that topic.

You can test the authentication method within the SQL server Management Studio by running the following syntax:

```
SELECT auth_scheme FROM sys.dm_exec_connections WHERE session_id = @@spid;
```

Preparing for the Client

This section describes how to prepare for the client.

Procedure: How to Harden the Java VM Cryptography

1. Obtain a copy of the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 8.
2. Unzip the archive and extract the following jar files to the location of the Java Virtual Machine runtime folder ({jre folder}/lib/security or {jre}\lib\security on linux, moving or replacing the current jars in the folder):

   - local_policy.jar
   - US_export_policy.jar

   Ensure that the correct runtime Java Virtual Machine folder is copied, as there may be multiple versions of java on a machine.

   If the jar files are not correct, the following error appears:

   ```
   Integrated Authentication Failed{guid number}
   ```

Procedure: How to Download the SQL Server Driver

1. Download the SQL Server driver version 6 from the Microsoft website at: *https://www.microsoft.com/en-us/download/details.aspx?id=11774*.

   For Windows machines, the extension is *exe*. For linux, a gzip *tgz*.

2. Unpack or install the driver into an appropriate folder.

Ticket-Granting Tickets for Kerberos

The KDC responds to an authentication service request of a client by returning a service ticket for itself. This special service ticket is called a ticket-granting ticket (TGT). A TGT enables the authentication service to safely transport the credentials of the requester to the ticket-granting service. It is meant only for use by the ticket-granting service.

The following list describes the main uses of a TGT:

An initial ticket from the authentication service of the user.

Used to request service tickets.

A Ticket Granting Ticket is similar to going through airport security, where you are validated that you are who you say you are. You can enter the secured area, but you will need something more to get on an actual flight.

Service Tickets

A service ticket enables the ticket-granting service (TGS) to safely transport the credentials of the requester to the target server or service. A service ticket is used to authenticate with services other than the TGS and is meant only for the target service.

A service ticket is what gets the credentials the client is providing to the target, in an encrypted format. The service ticket is a ticket to a particular computer or program.

A Ticket Granting Ticket is the very first thing needed, if this fails, nothing else happens. Once a client has a TGT, then a service ticket for a particular service can be requested and issued.

Using JAAS

The Java Authentication and Authorization Service is one of several Java modules that can be plugged for security. This allows applications to use security services without having to hard code values in the source code of a program. One of the plug-in modules is for Kerberos. A subject is a set of credentials representing a single entity. For a given subject, for example, a principal, representing the specifics of the type of subject, such as KRB_NT_PRINCIPAL, describes the formatting and representation of the credentials. A subject when passed to the Login module goes through states, (either {initialize, login, commit for success}, or {any error is initialized, login, abort, with logout the final stage for both}). There are multiple checks during Kerberos initialization. The Subject is usually the complete FQDN (Fully Qualified Domain Name) of the user ID and the Kerberos TGT ticket attached to it. The Principal is the FQDN of the user ID in the following user principal name format:

```
domainuser@REAL.COM
```

A keytab is a file containing pairs of Kerberos principals and encrypted keys (which are derived from the Kerberos password). You can use a keytab file to authenticate to various remote systems using Kerberos without entering a password. However, when you change your Kerberos password, you will need to recreate all your keytabs.

Keytab files are commonly used to allow scripts to automatically authenticate using Kerberos, without requiring human interaction or access to password stored in a plain-text file. The script is then able to use the acquired credentials to access files stored on a remote system.

Keytabs are used by the Java Authentication and Authorization Service for login to Kerberos.

The JAAS service uses different modules and parameters for plug-in security. The Kerberos module is called K5b5LoginModule and has the property *required* (there are properties for usage such as *optional* and so on). The Java Authentication and Authorization Service (JAAS) is used here solely for the purpose of authentication. For more information, see *Using JAAS* on page 239.

Creating a JAAS File for SQL Server Driver for Kerberos

The following syntax shows how to configure the SQL Server Driver to use the Krb5LoginModule, requiring its use and failing to start the driver if this login module fails. The debug mode is currently `ON debug=True`.

The login modules are called in a callback. Using `doNotPrompt=true` ensures that no screen texts appear asking for credentials, and fails if the credentials are not found. The Kerberos default ticket cache will not be use, as shown in `useTicketCache=false`. The Kerberos authentication key should be fetched and stored in memory, and the credentials are coming from the keytab. Optionally, the user principal can be typed in the JAAS file as `user@REALM1.REALM2.COM`, overriding the keytab. SQLJDBCDriver is a required term, on linux in lower case letters

```
SQLJDBCDriver {
  com.sun.security.auth.module.Krb5LoginModule required
    debug=true
   doNotPrompt=true
   useTicketCache=false
   useKeyTab=true
    keyTab="domainuser.keytab"
};
```

Placing JAAS Files and Keytabs in iWay Home Root

The keytab file and iwjaas.conf should be put into the iWay Service Manager root folder where the iWay7 or iWay8 command file is kept.

Windows JAAS File

```
SQLJDBCDriver {
  com.sun.security.auth.module.Krb5LoginModule required
//  refreshKrb5Config=false
   debug=true
   doNotPrompt=true
   useTicketCache=false
   useKeyTab=true
     keyTab="sqls6.keytab"
//      principal="user@REALM.COM";
```

Note: On Linux, *sqljdbc* must be in lowercase letters.

where:

`debug=true note that all values in the principalname MUST exist`

Turns on debug mode.

`doNotPrompt=true`

Does not prompt user for credentials.

`useTicketCache=false`

Does not use the Kerberos default cache.

`useKeyTab=true`

Uses a keytab instead of the cache.

`keyTab`

Is the Keytab name. Paths in this name do not always work.

`principal`

Overrides the principal in the keytab with this value. Note that all values in the principal name must exist.

Configuring Kerberos for Windows

The Windows machine must be located in the same domain where Kerberos is installed. The machine must have delegation enabled in the Domain Controller.

Modifying Windows Registry for Client Machines

You can modify the Windows registry to obtain a ticket, otherwise goes to the Local Security (LSA)

```
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Lsa\Kerberos\Parameters
Value Name: allowtgtsessionkey
Value Type: REG_DWORD
Value: 0x01
```

Setting Up Environmental Variables

To access Environmental Variables, right-click the Windows 10 *Start* menu, click *Control Panel*, select *System*, click *Advanced system settings*, and then click *Environmental Variables*.

You can set the following values as SYSTEM Variables:

```
KRB5_CONFIG = path to krb.ini
KRB5_TRACE = /dev/stdout
```

You must shut down and then start your Windows machine after making this change. Do not restart your machine.

Configuring the Kerberos Client

Perform the following steps for easier connection confirmation during testing:

1. Download and install the MIT Kerberos client for Windows from the MIT website:

   http://web.mit.edu/kerberos/dist/

2. When installation is complete, reboot your system.

   A Kerberos icon will appear on your desktop after restarting your machine.

3. Double-click the Kerberos icon to open Kerberos.

   If all server configurations are complete, a Kerberos ticket will be obtained and will appear in the dialog windows as *userid@domain.suffix* (user principal name format).

Procedure: How to Create a Keytab on Windows

The ktpass command is used to create a keytab from the Domain Controller only. To create a keytab:

1. Create a folder in a non-Windows namespace area.
2. Open a command shell and use the following syntax as a template:

   ```
   ktpass /princ myuser@REALM1.REALM2.COM /pass pass /ptype
   KRB5_NT_PRINCIPAL /crypto AES256-SHA1 /kvno 1 /out out.keytab
   ```

   Note: The principal is the name of the user ID where the service principal name is mapped.

where:

myuser

Is a real user name that you can use.

REALM1

Can be renamed to a real realm name and must only be one level or multiple levels deep.

pass

Is a real password that you can enter.

/out

The name of the keytab or name to use for the keytab. Do not use the /mapuser switch because it will overwrite the user principal name in the user account with what is entered after the switch. The principal name in the keytab must exactly match the real account credential name. If you skip this flag and ignore the warning, the Service Principal Name will carry the mapping between principal and user.

[-/] crytpo

Can be one of the following:

- DES-CBC-CRC: for compatibility
- DES-CBC-MD5: for compatibility
- RC4-HMAC-NT: default 128-bit encryption
- AES256-SHA1: AES256-CTS-HMAC-SHA1-96
- AES128-SHA1: AES128-CTS-HMAC-SHA1-96
- All: All supported types

Note: AES-128 or AES-256 are recommended for highest security. Check with the Active Directory administrator for the highest level supported.

3. Enter a real user name for *myuser* from the syntax above.

Downloading a Keytab to a Client Machine

You can map a drive using Windows Explorer and download the keytab to the windows client machine that will use the keytab. A keytab is not specific to a machine, but to a domain.

Configuring Kerberos for Linux

This section describes how to install the Kerberos Workstation for Linux.

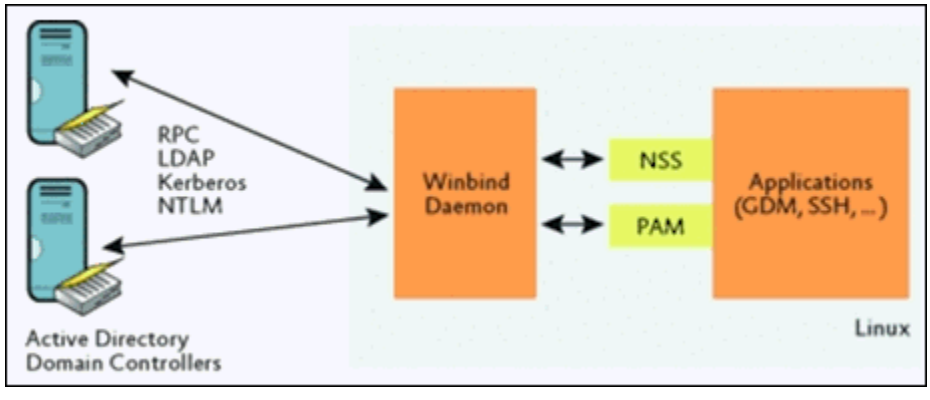
Procedure: How to Install Kerberos Workstation

1. Install the Kerberos Workstation files (not the server files) for the linux distribution using yum, dnf, yast, or another tool.

   The installation will install a krb5.conf file in the /etc folder. Some linux distros have a graphical Kerberos connect tool similar to Windows.

2. Connect the workstation to Active Directory using Samba and Windbind on the Linux OS.

   The Linux workstation must be connected to Active Directory, as shown in the following image.



   For more information on using Linux on Windows machines, see the Microsoft website at: *https://technet.microsoft.com/en-us/library/2008.12.linux.aspx*.

   For more information on configuring Samba, see the Samba website at: *https://www.samba.org/samba/docs/using_samba/ch03.html*.

   For more information on WinBind, see: *https://www.samba.org/samba/docs/man/Samba-HOWTO-Collection/winbind.html*.

Joining the Samba Server to the PDC Domain

All machines participating in domain security should be members of the domain. This applies also to the PDC and all BDCs.

The process of joining a domain requires using the *Net RPC* join command. This process communicates with the domain controller it registers with (usually the PDC) through *MS DCE RPC*. This means, of course, that the *SMBD* process must be running on the target domain controller. It is therefore necessary to temporarily start Samba on a PDC so that it can join its own domain.

Enter the following command to make the Samba server join the domain, where PDC is the name of your PDC and Administrator is a domain user who has administrative privileges in the domain.

Note Before attempting to join a machine to the domain, verify that Samba is running on the target domain controller (usually PDC) and that it is capable of being reached via ports 137/udp, 135/tcp, 139/tcp, and 445/tcp (if Samba or Windows Server 2Kx).

The following syntax shows the use of the Net RPC join facility:

```
root# /usr/local/samba/bin/net rpc join -S PDC -U Administrator
```

The proper response to the command is:

```
Joined the domain DOMAIN
```

where:

```
DOMAIN
```

Is your domain name.

Setting Environmental Variables as a Global Scope

The following syntax shows how to set the environmental variables as a global scope.

```
KRB5_CONFIG = path to /etc/krb5.conf
KRB5_TRACE = /dev/stdout
```

Creating a Keytab on Linux

To create a keytab:

```
type ktutil
ktutil: addent -password -p myuser@REALM1.REALM2.COM -k 1 -e aes256-cts-
hmac-sha1-96 (addentry)
ktutil: wkt /apps/kerberos/myuser.keytab (write keytab)
ktutil:quit
```

Procedure: How to Initialize a Keytab on Windows and Linux

Before initializing the keytab, make sure you are using Java Kerberos, since there are also MIT Kerberos, Microsoft Kerberos, and Heimdal Kerberos that may be installed on the machine. Ensure none of the others are in the PATH variable.

The Kinit program of Java can be found in the jre/bin folder.

On Windows and Linux, the following command line is the same:

```
kinit -V exampleuser@INNERREALM.OUTERREALM.COM -k -t exmpleuser.keytab
```

1. On Linux and Windows, type *klist tickets* to see any ticket obtained.
2. Type *klist tgt* to see information about the key server.
3. To start over for the kinit on Windows, type *klist purge*.
4. To start over for the kinit on Linux, type *kdestroy-A*.

```
Usage: kinit [-A] [-f] [-p] [-c cachename] [[-k [-t keytab_file_name]]
[principal] [password]
        available options to Kerberos 5 ticket request:
            -A   do not include addresses
            -f   forwardable
            -p   proxiable
            -c   cache name (i.e., FILE:\d:\myProfiles\mykrb5cache)
            -k   use keytab
            -t   keytab file name
            principal   the principal name (i.e., qweadf@ATHENA.MIT.EDU
qweadf)
            password   the principal's Kerberos password
Since the keytab already has a principal and password via ktutil or
ktpass, there is no need to enter them.
```

Creating Batch Jobs for Credential Expiration and Renewal on Windows and Linux

Depending on the Windows Domain settings, the credentials in a keytab expire in a short period.

Run the *kinit* command in a cron job on linux or a regularly scheduled service on Windows.

When the credentials expire, there is no error indications, but dialogs such as *Logging Exception*, *User Principal Invalid*, and several other messages may appear to mislead you.

Modifying the JAAS Configuration File

Modify the JAAS configuration file and enter the name of the current keytab in the filename.

Modifying the KRB5 Configuration File

Modify the krb5.conf file so it appears similar to the krb5.conf file found in *Kerberos Configuration File (krb5.conf)* on page 248.

On Windows, it is good practice to put krb5.ini in the \Windows folder. On Linux, it should be /etc/krb5.config.

Debugging Setup (optional)

For initial DEBUG setup, modify the logging.properties file in SQL Server driver installation directory, and add the following syntax:

```
com.microsoft.sqlserver.jdbc.level = ALL
```

Procedure: How to Configure iWay Service Manager

1. Start iWay Service Manager.
2. From the iWay console, click *Server* and then select *Java Settings*.
3. In the Property field, enter the following syntax:

   ```
   java.security.krb5.conf
   ```

4. In the Value field, enter the path and file name to the krb5.conf file, for example:

   ```
   c:\krb\krb5.conf
   ```

5. Click *Add*.
6. In the Property field, enter the following syntax:

   ```
   java.security.auth.login.config
   ```

7. In the Value field, enter the path and file name to the JAAS login configuration file, for example:

   ```
   C:\logon\logon.conf
   ```

8. Click *Add*.
9. In the Property field, enter the following syntax:

   ```
   sun.security.krb.debug
   ```

10. In the Value field, enter the following:

    ```
    true
    ```

11. Click *Add*.
12. After the setup is confirmed and working, come back and delete the last property.
13. In the iWay console, click *Providers* and then select *Data Provider*.
14. In the JDBC area, click *New*.
15. Enter the proper parameters for a SQL Server JDBC connection, with the proper JDBC connection string, as shown in the following syntax:

    ```
    jdbc:sqlserver://myserver/mydb.INNER.OUTER.COM:1433;databaseName=Fido;
    ```

16. You must end the connection string with the following syntax:

    ```
    integrated Security=true;authenticationScheme=JavaKerberos
    ```

17. Do not enter anything in the User or Password field.
18. Complete the rest of the form, and then click *Update*.
19. Stop and close iWay Service Manager, and then restart it.
20. Return to the JDBC driver screen, and then click the *Test* button to confirm that the connection is working.

    The Kerberos debug information will print in the iWay console window, You can find the domain name and SPN in the debug information, but all other data other than the Kerberos messages are encrypted.

    If the connection is OK, a Success message will print in the driver window.
21. If everything is ok, return to the Java Settings and remove the Kerberos debug setting.
22. If there are any errors, debug the Kerberos issues until the issues are resolved.

Kerberos Configuration File (krb5.conf)

Create a krb5.conf file and enter or change the following properties:

```
[logging]
 default = FILE:{location}
[libdefaults]
 default_realm = MY.DOMAIN.SUFFIX  (enter uppercase domain name)
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 allow_weak_crypto = true
# uncomment if compatibility problems arise
#default_tkt_enctypes = aes256-cts-hmac-sha1-96 aes128-cts des3-cbc-sha1
rc4-hmac #arcfour-hmac-md5 des-cbc-md5 des-cb
#c-crc
#default_tgs_enctypes = aes256-cts-hmac-sha1-96 aes128-cts des3-cbc-sha1
rc4-hmac #arcfour-hmac-md5 des-cbc-md5 des-cb
#c-crc
#permitted_enctypes = aes256-cts-hmac-sha1-96 aes128-cts rc4-hmac arcfour-
hmac-#md5
[realms]
MY.DOMAIN.SUFFIX {  (enter uppercase domain name)
 kdc = my.domain.suffix  (enter lowercase domain name)}
```

SQL Server Clustered Server Warning

It is recommended that you do not grant the WriteServicePrincipalName right to the SQL service account when the following conditions are true:

- There are multiple domain controllers.

- SQL Server is clustered.

In this scenario, the SPN for the SQL Server may be deleted because of latency in Active Directory replication. This may cause connectivity issues to the SQL Server instance.

Assume that you have the following:

- An SQL virtual instance named Sqlcluster with two nodes: Node A and Node B.
- Node A is authenticated by domain controller A, and Node B is authenticated by domain controller B.

The following may occur:

- The Sqlcluster instance is active on Node A, and registered the SQL SPN in domain controller A during start up.
- The Sqlcluster instance fails over to Node B when Node A is shutdown normally.
- The Sqlcluster instance deregistered its SPN from domain controller A during the shutdown process on Node A.
- The SPN is removed from domain controller A, but the change has not yet been replicated to domain controller B.
- When starting up on Node B, the Sqlcluster instance tries to register the SQL SPN with domain controller B. Since the SPN still exists, Node B does not register the SPN.
- After some time, domain controller A replicates the deletion of the SPN (from step 3) to domain controller B as part of Active Directory replication. The end result is that no valid SPN exists for the SQL instance in the domain and hence you see connection issues to the Sqlcluster instance.

Appendix F

WSO2 Identity Server Support

This section describes how to configure support for WSO2 Identity Server through iWay Service Manager (iSM).

In this appendix:

WSO2 Identity Server Introduction

The WSO2 Identity Server is an open source identity and entitlement management server with support for eXtensible Access Control Markup Language (XACML). This server can be leveraged by iWay Service Manager (iSM) to authenticate users and authorize access. For more information about the XACML specification, refer to the following website:

http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html

The WSO2 realm can validate users against a user store in WSO2 Identity Server. The list of roles is accessible in the principal returned by the realm.

The following iSM components can be used together to check whether access to a resource is authorized:

- XACML Provider
- XACML Service (com.ibi.agents.XDXacmlAgent)

iSM acts as a Policy Enforcement Point (PEP) calling an external Policy Decision Point (PDP) running in WSO2 Identity Server.

These two features can be used independently or together. For example, it is possible to use the WSO2 realm and authorize access explicitly by testing the roles in the principal. Conversely, it is possible to use any iSM user realm and authorize access with XACML.

Installing and Configuring WSO2 Identity Server

This section describes how to install and configure WSO2 Identity Server.

Procedure: How to Install WSO2 Identity Server

1. Download the WSO2 Identity Server binary distribution from the following website:

   http://wso2.com/products/identity-server

   The WSO2 Identity Server binary is packaged as a .zip file (wso2is-4.1.0.zip).

2. Unzip the wso2is-4.1.0.zip file to a directory on your file system.

3. To run WSO2 Identity Server, navigate to the <*WSO2_Home*>\bin directory and run the wso2server.bat file.

   Note: By default, WSO2 Identity Server uses port 9999 for the JMXServerManager. This conflicts with the default console port that is used by the iSM base configuration. For a quick workaround, you can start iSM before WSO2 Identity Server, which will start the WSO2 Identity Server without JMX. For a more permanent solution, change the console port of the iSM base configuration or can change the RMIRegistryPort setting in the carbon.xml file, which is located in the <*WSO2_Home*>\repository\conf directory. For example:

   ```
   <RMIRegistryPort>9999</RMIRegistryPort>
   ```

4. To stop WSO2 Identity Server, type `Control-C` in the command prompt window and press Enter.

5. To open the WSO2 Identity Server console, enter the following URL in your browser:

   ```
   https://localhost:9443/carbon/
   ```

   Note: The browser will generate a message, which indicates that the certificate is not valid. You can ignore this message and continue. In addition, the menu might not appear correctly if you are using a Microsoft Internet Explorer Version 10 browser. However, it looks better using a Google Chrome browser.

6. Enter *admin* for the user name and password to log on as the default administrator account.

Procedure: How to Extract and Import the SSL Certificate for WSO2 Identity Server

The Secure Sockets Layer (SSL) certificate of WSO2 Identity Server is stored in the KeyStore defined by the <KeyStore> entry in the carbon.xml file, which is located in the <*WSO2_Home*> \repository\conf directory. For example:

```
<KeyStore>
   <Location>${carbon.home}/repository/resources/security/wso2carbon.jks
   </Location>
   <Type>JKS</Type>
   <Password>wso2carbon</Password>
   <KeyAlias>wso2carbon</KeyAlias>
   <KeyPassword>wso2carbon</KeyPassword>
</KeyStore>
```

The default WSO2 SSL certificate is self-signed. Since there is no Certificate Authority (CA), you need that certificate itself in your TrustStore to validate it. This explains the warning message generated in the browser when accessing the console because the self-signed certificate is not in the browser trust store.

For a quick test, you can use the wso2carbon.jks KeyStore itself as a TrustStore. However, a better approach would be to extract the certificate by using the following command:

```
keytool -exportcert -alias wso2carbon -file wso2carbon.cert -keystore wso2carbon.jks -
storetype JKS -storepass wso2carbon
```

Then import the certificate within an empty KeyStore or an existing TrustStore by using the following command:

```
keytool -importcert -trustcacerts -alias wso2carbon -file wso2carbon.cert -keystore
wso2ts.jks -storetype JKS -storepass wso2password
```

This is the technique that was used to produce the wso2ts.jks KeyStore. For example:

```
iwcore/test/data/providers/keystores/wso2ts.jks
```

The best solution is to create a real SSL server certificate for WSO2 Identify Server and configure it in the carbon.xml file. Then you would put the CA of that certificate in our TrustStore. This is only required for production environments.

Procedure: How to View WSDL Files of Web Services Implemented in WSO2 Identity Server

To view the WSDL files of web services implemented in WSO2 Identity Server, you must first instruct the server to unhide these files.

1. Edit the carbon.xml file, which is located in the <*WSO2_Home*>\repository\conf directory.
2. Change the *HideAdminServiceWSDLs* setting to *false*. For example:

   ```
   <HideAdminServiceWSDLs>false</HideAdminServiceWSDLs>
   ```

   The description of the web services called by iSM can be viewed using the following URLs:

   ```
   https://localhost:9443/services/RemoteUserStoreManagerService?wsdl
   ```

```
https://localhost:9443/services/EntitlementService?wsdl
```

Procedure: How to Debug WSO2 Identity Server

To debug WSO2 Identity Server, you must enable log4j debugging by using one of the following options:

1. Edit the log4j.properties file, which is located in the <*WSO2_Home*>\repository\conf directory.
2. In the WSO2 Identity Server console, click *Configure*, select *Logging*, and change any required loggers to the TRACE level.

Note: Modifying these logging settings using the WSO2 Identity Server console saves the settings in the repository, but not in the log4j.properties file. The repository overrides what is defined in the log4j.properties file.

Configuring WSO2 Users and Roles

If you use WSO2 Identify Server only to authorize access with XACML, then there is no need to create users and roles. You can proceed directly to *Configuring XACML Policies* on page 254.

Procedure: How to Create a New User

1. Click *Configure* in the left pane, followed by *Users and Roles*, *Users*, and then *Add New User*.
2. Enter a user name and password for the new user.
3. Click *Next*, select the existing roles you want to assign to the new user and then click *Finish*.

Procedure: How to Create a New Role

1. Click *Configure* in the left pane, followed by *Users and Roles*, *Roles*, and then *Add New Role*.
2. Enter a name for the new role and then click *Next*.
3. Since the WSO2 permissions are not relevant to iSM users, click *Next*.
4. Enter a regular expression to display a list of users.
5. Select which existing users belong to that role and then click *Finish*.

Configuring XACML Policies

If you use WSO2 Identify Server only to authenticate users, then there is no need to create XACML policies.

To create an XACML policy from scratch, use the simple editor or the advanced editor.

Procedure: How to Create XACML Policies

1. Click *Main*, followed by *Administration*, and then *Add New Entitlement Policy*.
2. You can import an XACML policy by clicking *Main*, followed by *Administration*, and then *Import New Entitlement Policy*.
3. Choose *File System*.
4. Click *Choose File* to open the dialog that allows you to select the file.
5. Click *Upload*.

   There are two policy files that can be imported for testing:

   ```
   components\iwcore\test\data\providers\xacml\policy1.txt
   ```

   ```
   components\iwcore\test\data\providers\xacml\policy2.txt
   ```

6. Once imported, click *Promote to PDP* next to each policy.

   WSO2 Identity Server Version 4.1.0 accepts the XACML Version 3 schema. You will receive an error if you try to import a policy written for the XACML Version 2 schema. Ensure that you have the correct version if you experiment with a sample policy you found on the web.

   The XACML engine inside WSO2 Identity Server is called *Balana*.

Procedure: How to Debug XACML Policies

To debug an XACML policy, you must enable log4j debugging by using one of the following options:

1. Edit the log4j.properties file, which is located in the <*WSO2_Home*>\repository\conf directory.
2. In the WSO2 Identity Server console, click *Configure* in the left pane, select *Logging*.

   You want to turn on TRACE level for loggers with the word *entitlement* and/or the word *balana* in the name.

Configuring iWay Service Manager

The following list outlines the configuration steps that are required in iWay Service Manager (iSM).

1. Configure a KeyStore Provider to serve as the TrustStore.

The KeyStore file must contain the root Certificate Authority (CA) that signed the Secure Sockets Layer (SSL) certificate used by WSO2 Identity Server. By default, the SSL certificate is self-signed, so the root CA is the SSL certificate itself.

2. Configure an SSL Context Provider.

   Select the KeyStore Provider that was configured in step 1 for the TrustStore. The KeyStore can remain empty. The Security Protocol is TLS or higher.

3. Configure an HTTP Client Provider.

   Select the SSL Context Provider that was configured in step 2 for the SSL Context

4. Continue the iSM configuration with the remaining topics in this section that are applicable.

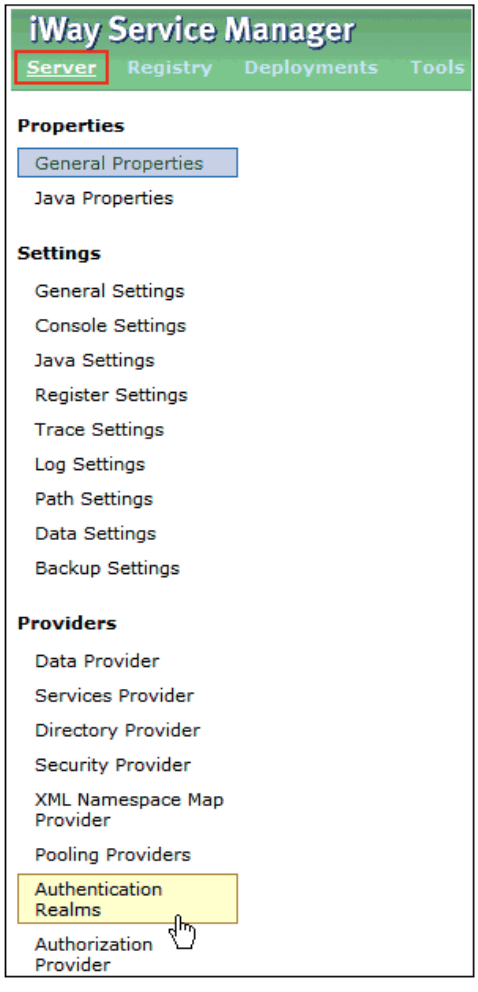
For more information on how to configure a KeyStore Provider, SSL Context Provider, and HTTP Client Provider, see the *iWay Service Manager User's Guide*.

Configuring an Authentication Realm for WSO2 Identity Server

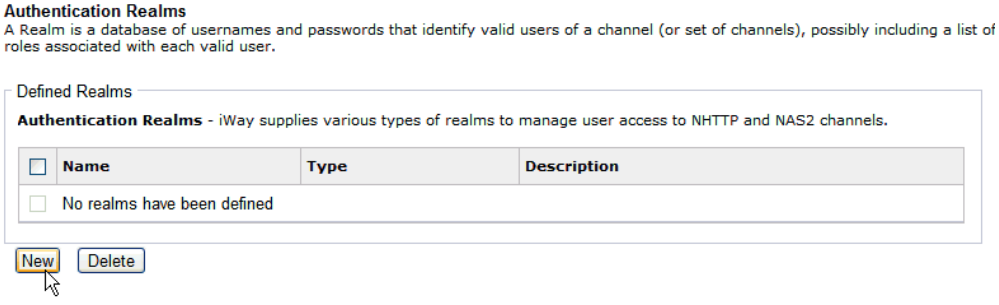
If you use WSO2 Identify Server only to authorize access with eXtensible Access Control Markup Language (XACML), then there is no need to create a WSO2 realm. You can proceed directly to *Configuring the XACML Provider and XACML Service* on page 263.

Procedure: How to Configure an Authentication Realm for WSO2 Identity Server

1. In the iWay Service Manager Administration Console, click *Server* in the top pane, and then click *Authentication Realms* in the left pane, as shown in the following image.

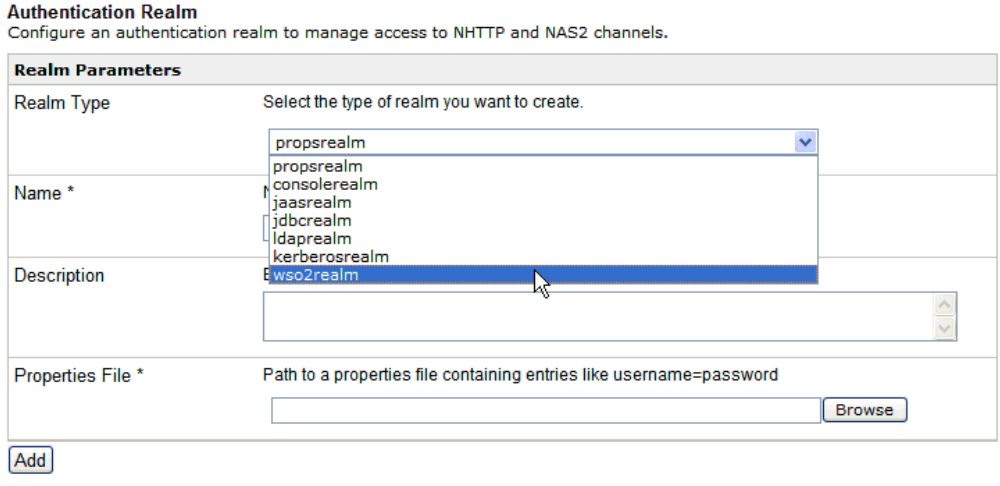


The Authentication Realms pane opens, as shown in the following image.



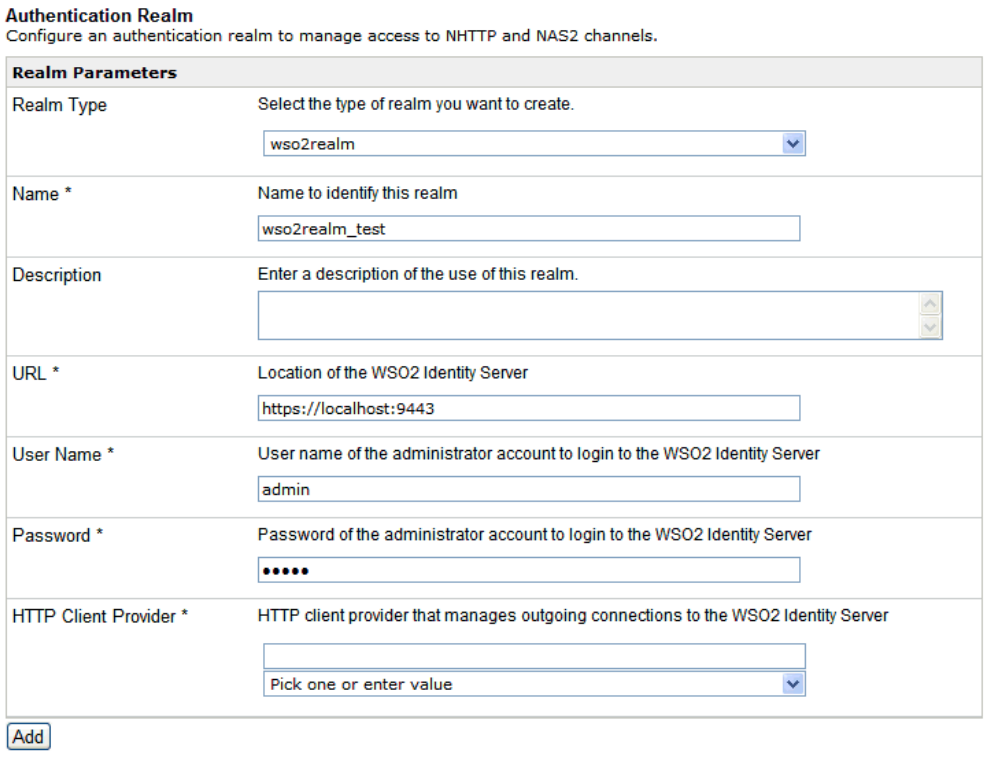
2. Click *New*.

   The Authentication Realm pane opens, as shown in the following image.



3. From the Realm Type drop-down list, select *wso2realm*.

The Authentication Realm pane is refreshed for the specific type of realm (in this case, WSO2), as shown in the following image.



4. Enter a name for the new realm you are creating (for example, wso2realm_test).
5. Enter the name of the Provider and the location of the WSO2 Identity Server.
6. Enter the URL where WSO2 Identity Server can be accessed.
7. Specify the user name and password for the administrator account in WSO2 Identity Server.

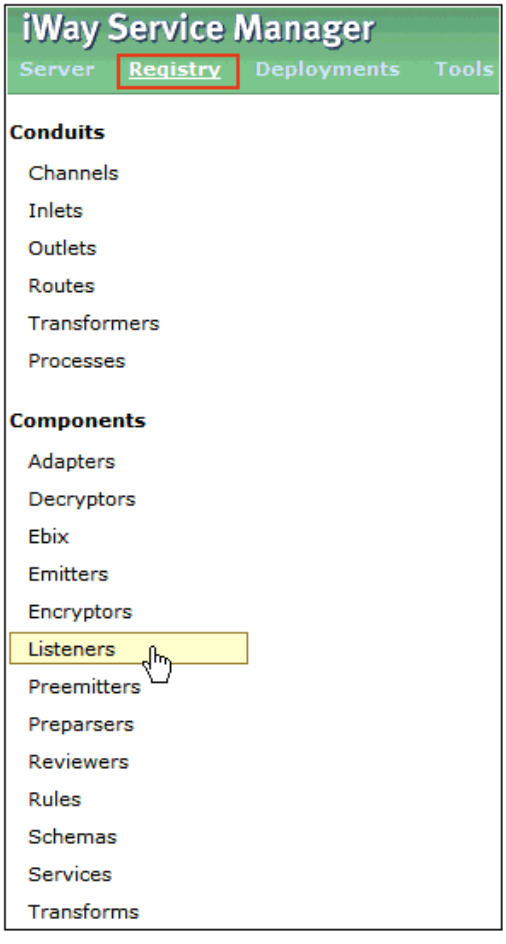
   The default user name and password is *admin*.

   This account is used to login to WSO2 Identity Server through HTTP Basic Authentication. The password is sent essentially in clear text, but the connection is using SSL.
8. From the HTTP Client Provider drop-down list, select the HTTP Client Provider that you defined earlier for the HTTP Client.
9. Click *Add*.

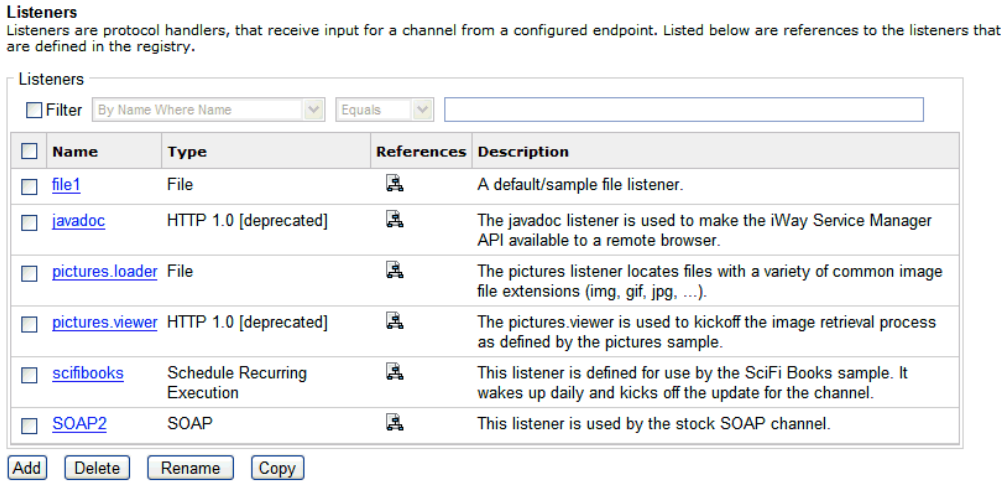
Procedure: How to Configure a Listener

Configure a listener that is realm-aware (for example, the NHTTP listener).

1. In the iWay Service Manager Administration Console, click *Registry* in the top pane, and then click *Listeners* in the left pane, as shown in the following image.

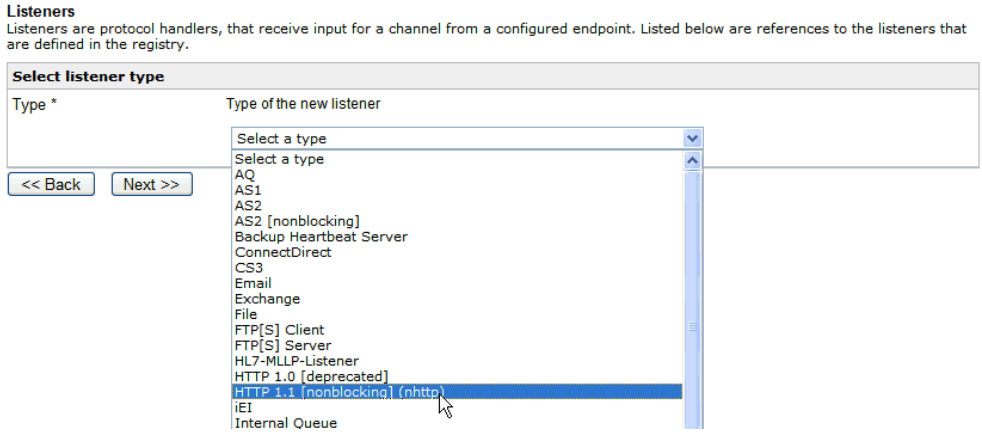


The Listeners pane opens, as shown in the following image.



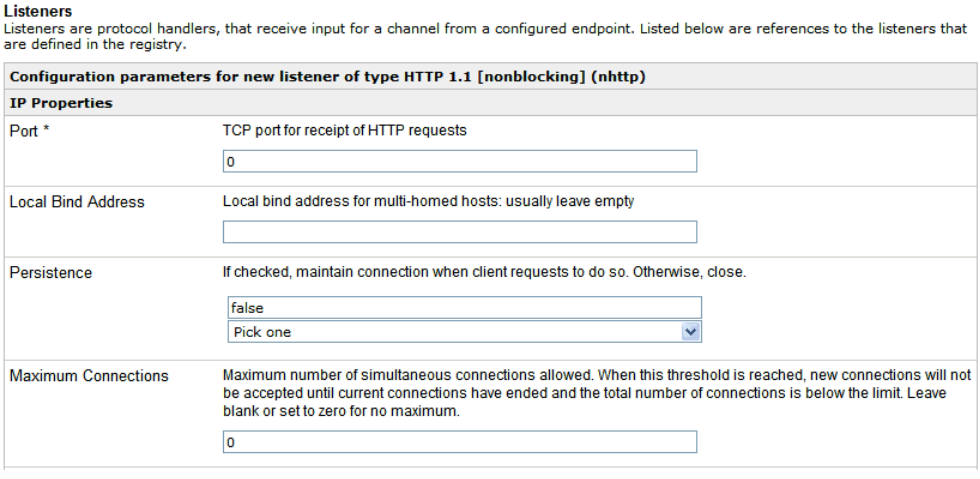
2. Click *Add*.

   The Select listener type pane opens, as shown in the following image.

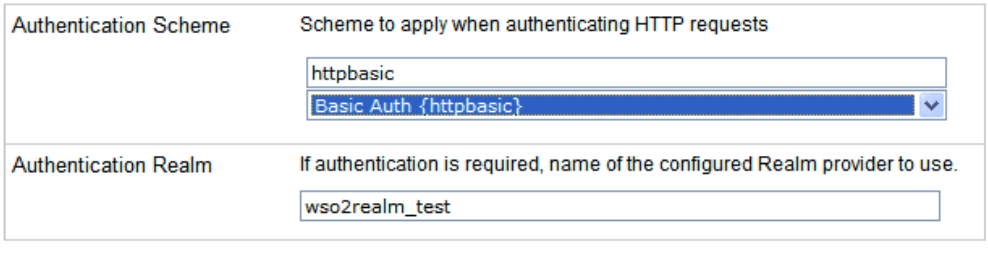


3. Select *HTTP 1.1 [nonblocking] (nhttp)* from the Type drop-down list and click *Next*.

The configuration parameters for the NHTTP listener are displayed.



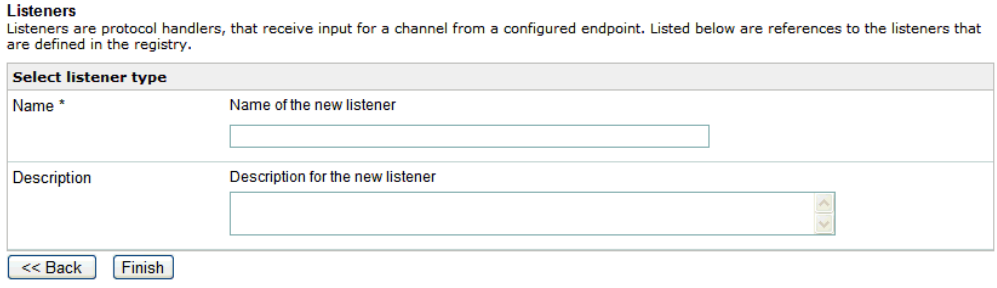
4. Specify a port where HTTP requests will be received.
5. Scroll down to the Authentication Realm parameter and enter the name of the WSO2 realm that was configured in the iSM Administration Console.



6. From the Authentication Scheme drop-down list, select *Basic Auth {httpbasic}*.

   Note: *Basic Auth* is insecure unless it operates under HTTPS.
7. Click *Next* at the bottom of the page to continue.

A listener name and description pane opens, as shown in the following image.



8. Enter a name for the selected listener and a brief description (optional).
9. Click *Finish*.

When this listener is deployed as part of a channel, it will ask for a user name and password, which will be checked against the user store within WSO2 Identity Server.

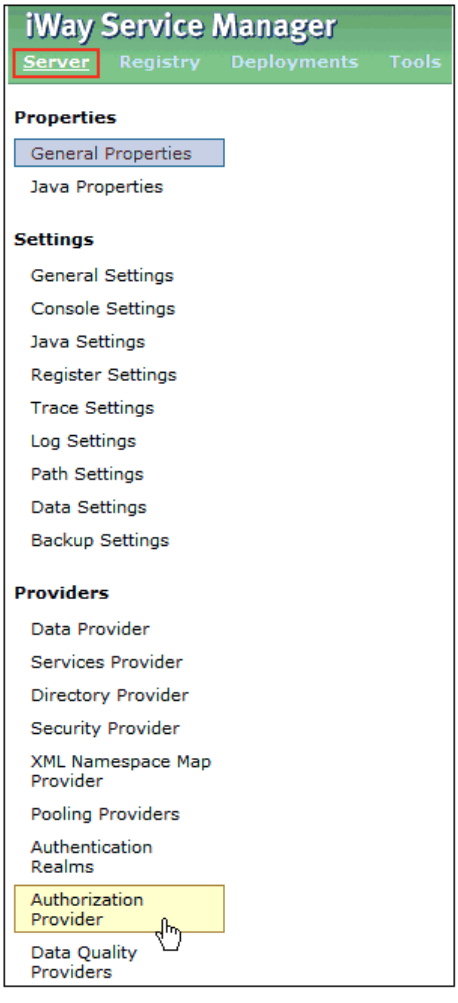
Configuring the XACML Provider and XACML Service

If you use WSO2 Identify Server only to authenticate users, then there is no need to create an XACML Provider or an XACML service.

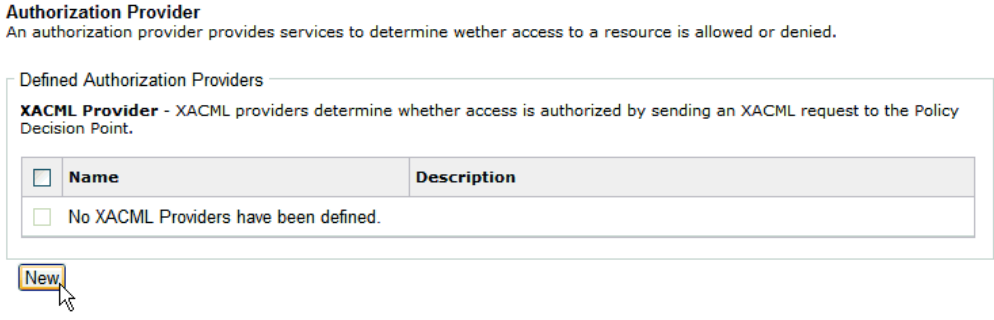
Procedure: How to Configure the XACML Provider

The XACML Provider helps centralize the XACML Policy Decision Point (PDP) configuration. It can also be used later as the site of a future XACML cache.

1. In the iWay Service Manager Administration Console, click *Server* in the top pane, and then click *Authorization Provider* in the left pane, as shown in the following image.

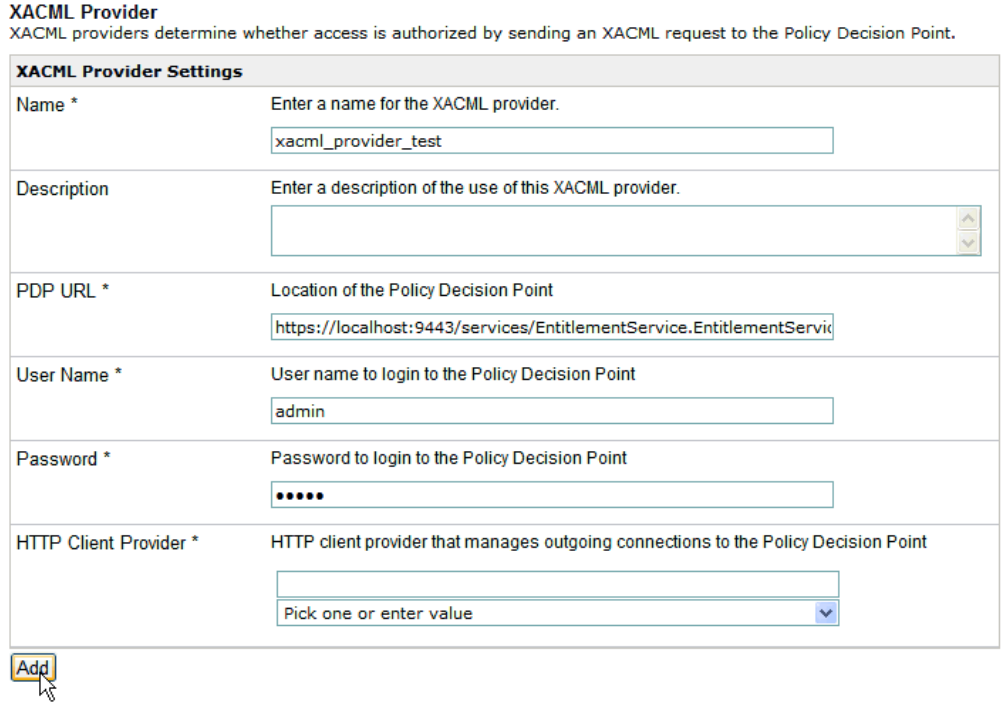


The Authorization Provider pane opens, as shown in the following image.



2. Click *New*.

   The XACML Provider pane opens, as shown in the following image.



3. Enter a name for the new XACML Provider you are creating (for example, xacml_provider_test).
4. In the PDP URL field, the default service location for the server is entered as follows:

```
https://localhost:9443/services/
EntitlementService.EntitlementServiceHttpsSoap11Endpoint/
```

Currently, the XACML provider assumes that the service location accepts SOAP 1.1.

5. Specify the user name and password for the administrator account in WSO2 Identity Server.

   The default user name and password is *admin*.

6. From the HTTP Client Provider drop-down list, select the HTTP Client Provider that you defined earlier for the HTTP Client.

7. Click *Add*.

Procedure: How to Configure the XACML Service

1. Create a process flow and add a Service object that points to XACML Service (com.ibi.agents.XDXacmlAgent).

2. Enter values for the XACML Service parameters as listed and described in the following table.

Parameter	Description
Subject	Determines who is requesting access to the resource. Enter the following: `enter _getprin('user')` This assumes the process flow is running under a listener configured with an authentication realm. This function returns the name of the logged in user, which is taken from the current principal.
Resource	Enter the name of the resource for which you wish to authorize access.
Action	Enter the type of action you wish to authorize (for example, *read*). **Note:** The Resource name and the Action is arbitrary, but it must be agreed with the XACML policy author.
XACML Provider	The name of the XACML Provider you configured earlier (for example, xacml_provider_test), which is used to send the XACML request.

3. Save the settings for the Service object for XACML Service (com.ibi.agents.XDXacmlAgent).

The XACML Service returns as *success* if the Policy Decision Point (PDP) returns *Permit* and *fail_security* otherwise. The actual decision from the PDP is available in the *xacml_decision* Special Register (SREG) if there is a need to distinguish *Deny*, *NotApplicable*, or *Indeterminate*.

The XACML Service calls the EntitlementService of the WSO2 Identity Server.

A sample request document can have the following structure:

```
<env:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ws="http://org.apache.axis2/xsd">
       <env:Body>
          <ws:getDecisionByAttributes>
             <ws:subject>user1</ws:subject>
             <ws:resource>http://localhost:9999/resource1</ws:resource>
             <ws:action>read</ws:action>
          </ws:getDecisionByAttributes>
       </env:Body>
    </env:Envelope>
```

Within WSO2 Identity Server, this is mapped to an XACML request document, which has the following structure:

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
ReturnPolicyIdList="false" CombinedDecision="false">
       <Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject">
          <Attribute IncludeInResult="false"
AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id">
             <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">user1</AttributeValue>
          </Attribute>
       </Attributes>
       <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:resource">
          <Attribute IncludeInResult="false"
AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id">
          <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">http://localhost:9999/resource1
          </AttributeValue>
          </Attribute>
       </Attributes>
       <Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:action">
          <Attribute IncludeInResult="false"
AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id">
             <AttributeValue DataType="http://www.w3.org/2001/
XMLSchema#string">read</AttributeValue>
          </Attribute>
       </Attributes>
    </Request>
```

A sample response document from the EntitlementService can have the following structure:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
       <soapenv:Body>
          <ns:getDecisionByAttributesResponse xmlns:ns="http://
org.apache.axis2/xsd">
             <ns:return><![CDATA[<Response xmlns="urn:oasis:names:tc:xacml:
3.0:core:schema:wd-17">
    <Result>
    <Decision>Deny</Decision>
    <Status>
    <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
    </Result>
    </Response>]]></ns:return>
          </ns:getDecisionByAttributesResponse>
       </soapenv:Body>
    </soapenv:Envelope>
```

Notice the XACML response is returned as a string, and not as embedded XML.

Developer Notes

To view the names of all web service names implemented by WSO2 Identity Server, enter the following command to start the server with the OSGi console enabled:

```
wso2server.bat –DosgiConsole
```

The black console will now show a prompt and accept commands. Type help to see a list of supported commands. To view the list of web services, type listAdminServices.

A long list of web service names is displayed.

Appendix G

User-Defined Permissions and Roles

This section describes how to configure the user-defined permissions and roles.

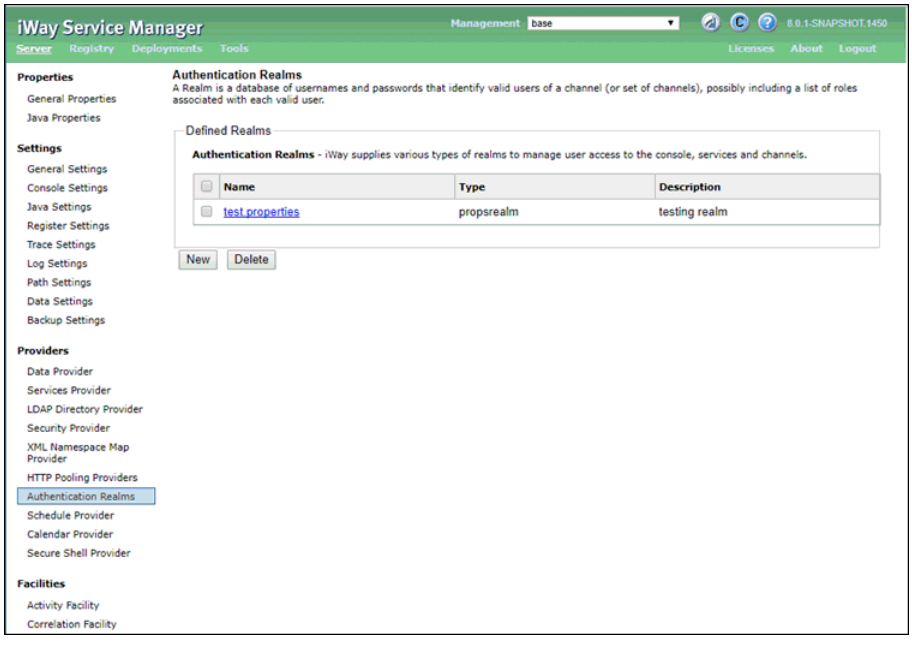
User-defined permissions enable an application to validate authority to perform an action in a flow. In a process flow or an agent configuration, an iFL statement can check for the presence of the named permission and take action accordingly.

For example, permissions can be assigned to a role that allows a specific source of message (which contains its own credentials that are validated through the XDPrincipal agent) to take action in the flow depending on the permission assigned to the credentials.

In this appendix:

Using Realms, Roles, and Permissions

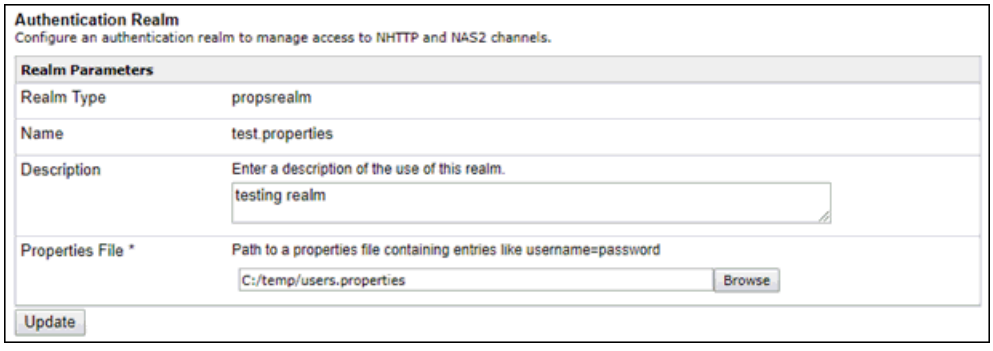
Used in conjunction with the Authentication Realm and console settings of Service Manager, the Security officer can allocate and restrict access of the user to the Service Manager toolsets based on roles and role assignment to users. The following example uses the simplest propsrealm for illustration.



Using Propsrealm

The propsrealm is short for the properties realm. It is a properties file that contains the security information that the Security Officer sets up to control access to iWay Service Manager.

The following image shows the realm parameters for propsrealm.



Enter the name and location of the security properties file, and a brief description of the realm.

Using the Security Property File

The security properties file is used by propsrealm to authenticate users and grant those users access to iWay Service Manager functions.

The following image is an example of a security properties file.

```
webSphere_service@ibi.com=websphere_service
webSphere_service@ibi.com.role0=domain mgm
pgmtst2@ibi.com=pgmtst2
pgmtst2@ibi.com.role0=pgmgrp1.admin
pgmtst4@ibi.com=pgmtst4
pgmtst4@ibi.com.role0=pgmgrp2.admin
kc05418=mypass
kc05418.role0=ism.admin
aw01@ibi.com=aw01
aw01@ibi.com.role0=COR-IT-BIADMIN
aw01@ibi.com.role1=COR-IT-BISUPPORT
aw01@ibi.com.role2=COR-IT-REMOTEACCESS
aw01@ibi.com.role3=aw01
cw01@ibi.com=cw01
cw01@ibi.com.role0=cw01
```

The file contains the following users:

```
WebSphere_service@ibi.com
pgmtst2@ibi.com
pgmtst4@ibi.com
kc05418
kc05418.role0
aw01@ibi.com
cw01@ibi.com
```

Each user in the properties file has a password followed by any number of roles (labeled *role0* through *roleN*) that the Security Officer assigns.

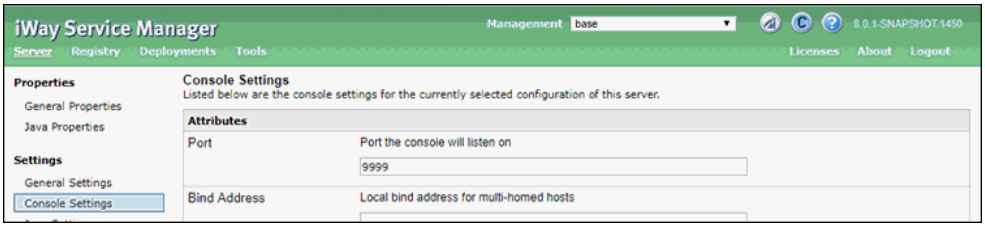
For more information on roles, see Server Roles [XREF].

Using the Realm

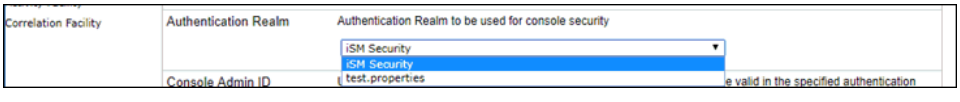
Do not change the console realm if your Security Officer has not set up all roles and permissions before this point. If everything is in place (all roles and permissions assigned), you will be ready to replace the Authentication Realm of the console.

To use the newly created propsrealm, it will have to be configured in the Console Settings.

1. Under Settings on the left pane of Service Manager, click *Console Settings*, as shown in the following image.



2. From the Authentication Realm drop-down list, select the propsrealm that you created, as shown in the following image.



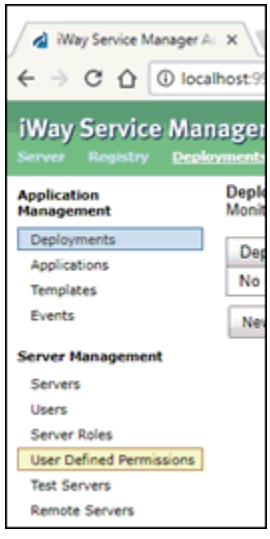
3. Stop and restart Service Manager for the changes to take effect.

Procedure: How to Create User-Defined Permissions

To create user-defined Permissions:

1. Click *Management* on the Service Manager home page, as shown in the following image.

2. Under Server Management on the left menu pane, click *User Defined Permissions*, as shown in the following image.



A list of all currently defined User Permissions appears.

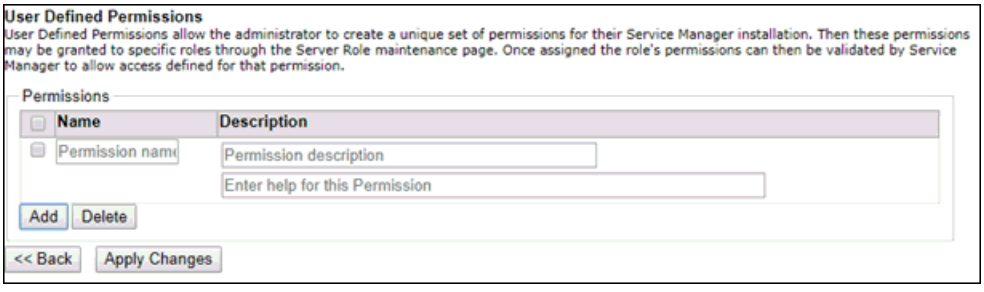
The following image shows no permissions defined.

Procedure: How to Add a User-Defined Permission

To add a User Defined Permission to iWay Service Manager:

1. Click the *Add* button.

The new permission is added to the end of the current listing. The following image shows only the new permission added. No other permissions have been defined.

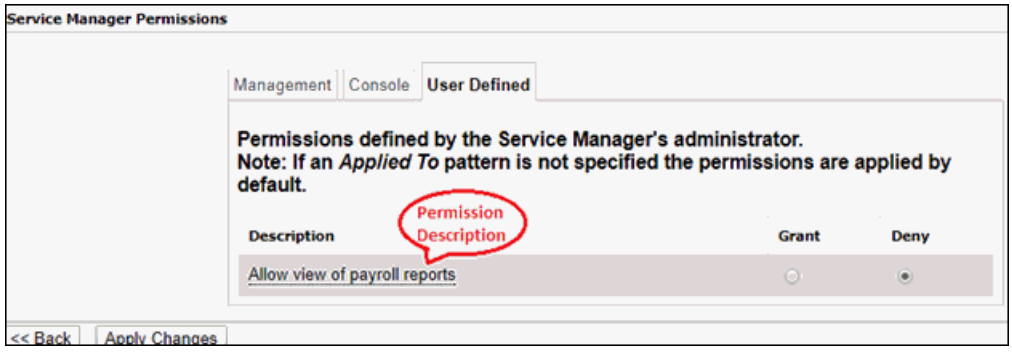


The following table lists and describes the field parameters.

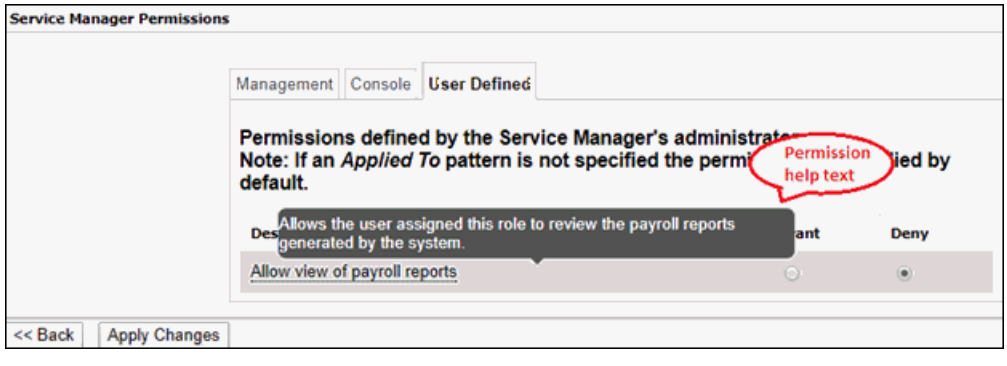
Parameter	Description
Permission Name	The Permission Name field must consist of a lower case string of characters that should not contain any imbedded whitespace characters (for example, spaces, tab, and so on). The name does not necessarily need to describe the permission (for example, allowview, accessforma, viewemppayroll).
Permission Description	The Permission Description field allows you to provide a detailed explanation of the permission. This explanation will be displayed later in the Server Roles management page. The Permission Description can be phrases such as *Allow view of payroll*, *Allow update of payroll*, or *View profit report*. However, the Permission description must be detailed enough to get the permissions scope understood.

Parameter	Description
Enter help for this Permission	The *Enter help for this Permission* field allows you to provide a more detailed description of the scope of the permission. This description will be displayed later in the Server Roles management page when the mouse cursor is hovered over the permission description.

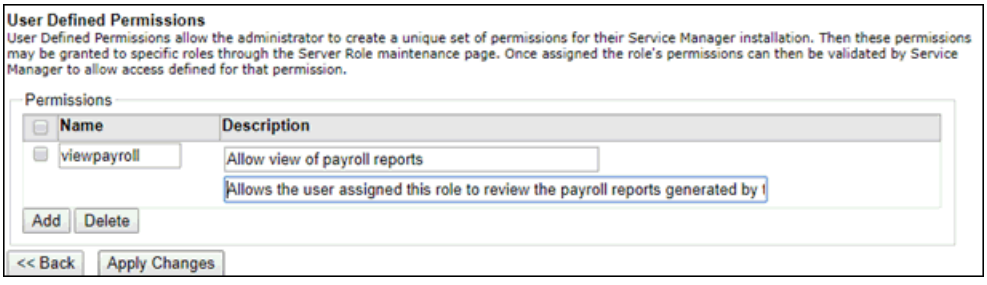
The following image shows the Permission Description.



The following image shows the Server Role page displaying the *Enter Help for this Permission* tool tip.



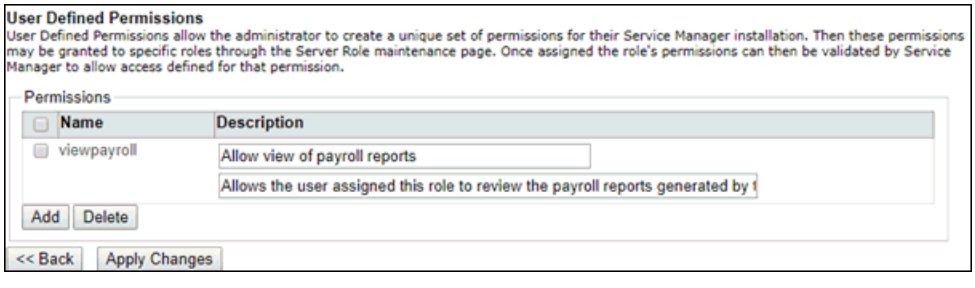
2. Click *Apply Changes* when you are satisfied that the permission name, description, and help text are complete, as shown in the following image.



If you do not click the Apply Changes button, or exit the screen by selecting another menu option or clicking *Back*, any changes made will be lost.

Updating Permissions

When the changes have been applied to iWay Service Manager, you will no longer be able to change the name of the permission. The only fields that may be changed are the *Permission Description* and the *Enter help for this Permission*, as shown in the following image.

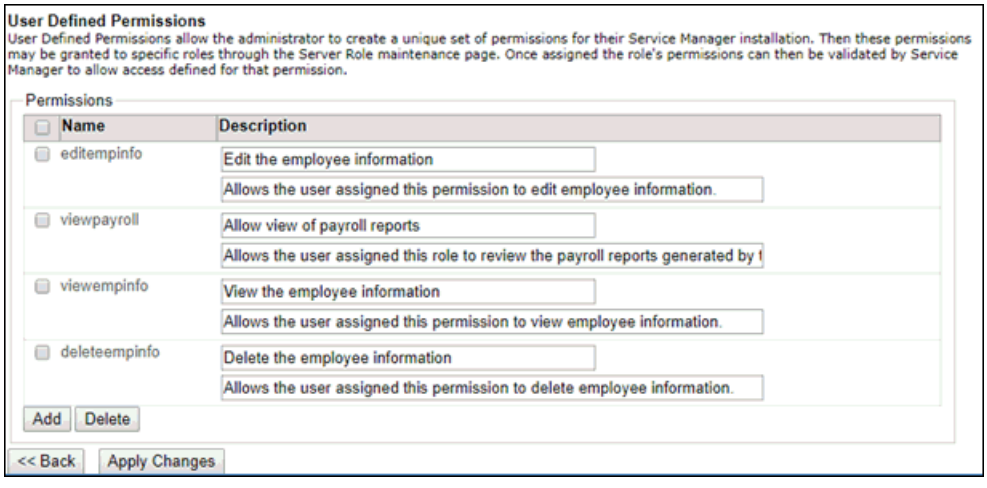


Once you have completed entering the permission description and the permission help text, click *Apply Changes*.

Similar to adding a permission, if you do not click the *Apply Changes* button and instead exit the screen either by selecting another menu option or by clicking the Back button, any changes made will be lost.

Deleting a Permission

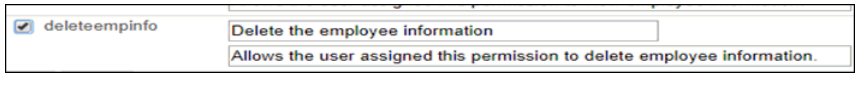
In general, you can click the *Delete* button to delete permissions permanently from Service Manager if a single permission or several permissions become obsolete.



You can delete a single permission by selecting the check box preceding the permission name. For example, the *Delete the employee information*(*deleteempinfo*) is no longer a stand-alone permission, but has been combined into the *Edit the employee Information* (*deleteempinfo*) permission.

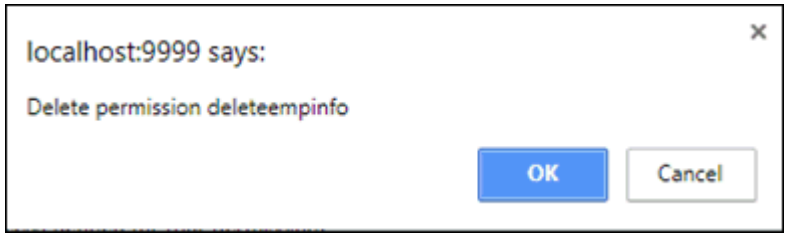
To delete the *deleteempinfo* permission:

1. Select the *deleteempinfo* permission check box, as shown in the following image.



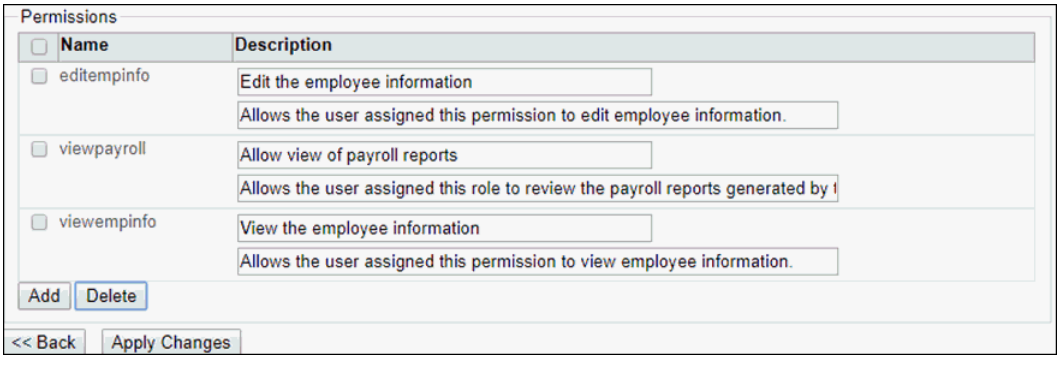
2. Click the *Delete* button.

   The following confirmation dialog box appears:



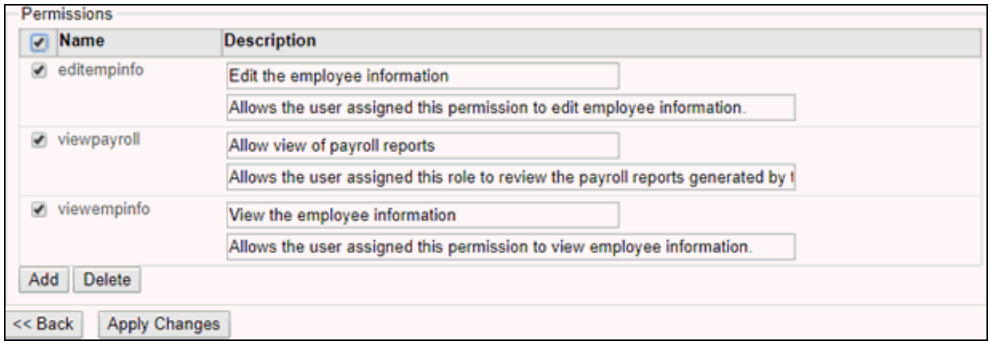
3. Click *OK* to confirm the deletion of the permission from the list.

The permission no longer appears in the list of permissions, as shown in the following image.



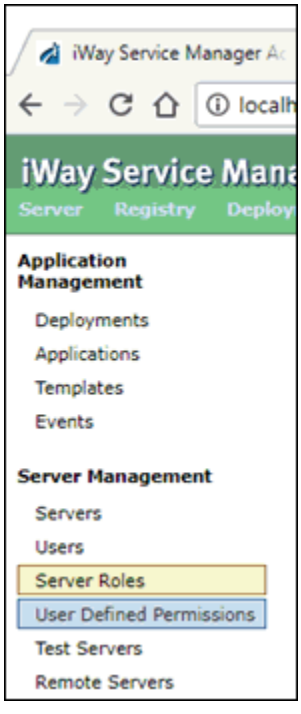
Similar to the Add and Update functions, any deletions are not finalized until you click *Apply Changes*. When a permission is deleted, *any* roll with that permission granted will have that permission removed.

Note: Selecting the check box at the top of the list in the heading row of the table either selects all check boxes or clears them from the permission list. The following image shows the heading check box selected, in order to select all permissions from the list.

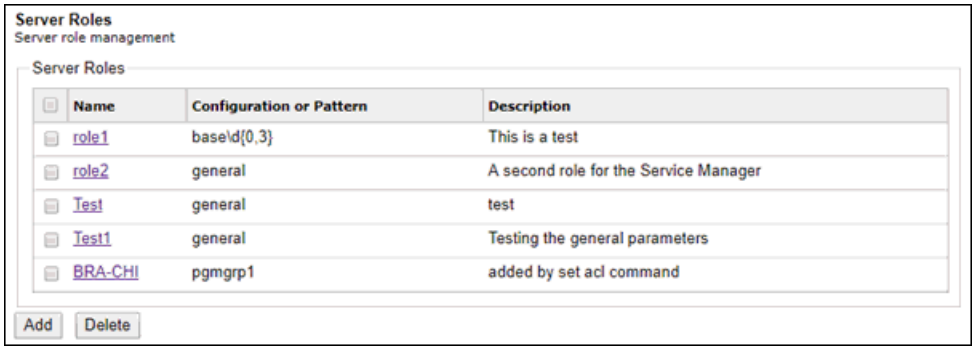


Server Roles

Under the Server Management section on the left pane, click *Server Roles*, as shown in the following image.



The Server Roles page appears, displaying a list of all currently crated roles for iWay Service Manager, as shown in the following image.

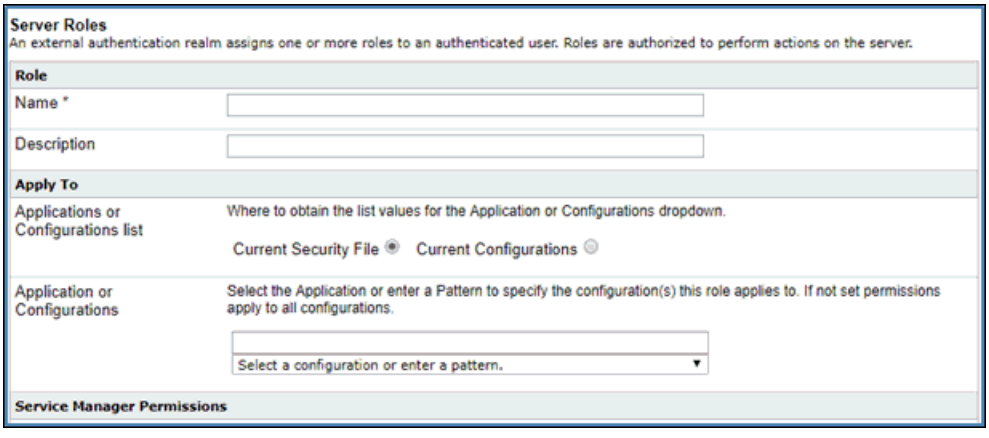


Procedure: How to Add Roles

To add a new Server Role to iWay Service Manager:

1. From the Server Roles page, click *Add*.

   The Server Roles maintenance pane appears, as shown in the following image.



The following table lists and describes the parameters of the maintenance pane.

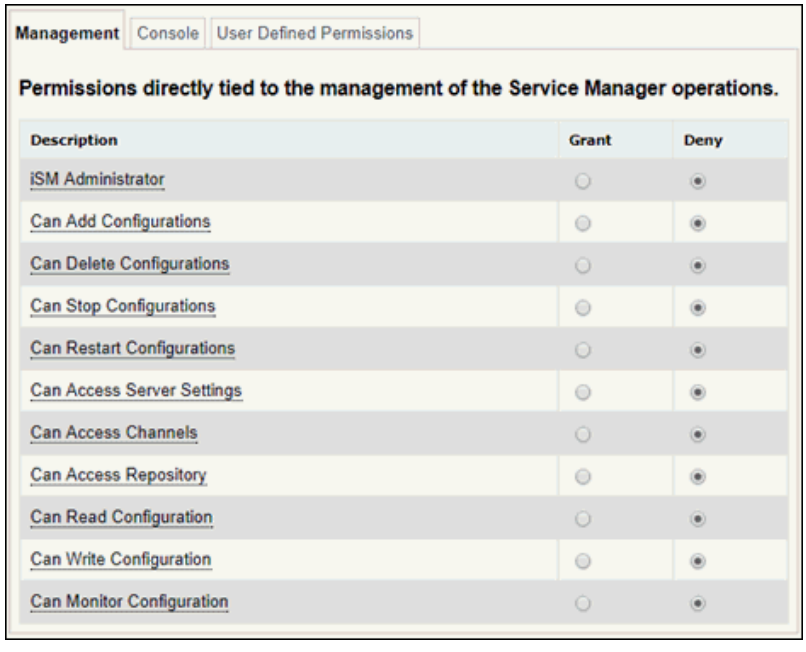
Parameter	Description
Name	Enter a name for this role. This name can be used in the following iFL function to verify that the user has this specific role: `_hasrole(name)`
Description	Enter a description that will allow you to know what the role is used for while viewing the list of Server Roles. Be as descriptive as possible.

Parameter	Description
Apply To	This field is used to identify the scope of the role. It is further subdivided into the following sections: ❑ **Applications or Configurations list.** This section allows the Security officer to select a value from scopes currently defined in the Security file, or select a value from current configurations and applications. Selecting the *Current Security File* radio button loads the Applications or Configuration drop-down list with values that currently have been implemented in the Security file. Selecting the *Current Configuration* radio button loads the Applications or Configuration drop-down list with Configurations and Applications that this instance on the Service Manager controls. ❑ **Application or Configuration.** This field defines the scope of influence of the role. A blank entry in this field indicates that the scope of the roll is *general*. The *general* scope is applied to a role if the role does not have a specific permission within the scope that it was defined. You can either select an entry from the drop-down list of values which contain the currently defined Service Manager configurations, applications, and patterns, or enter a new Regular Expression pattern for this role to use.

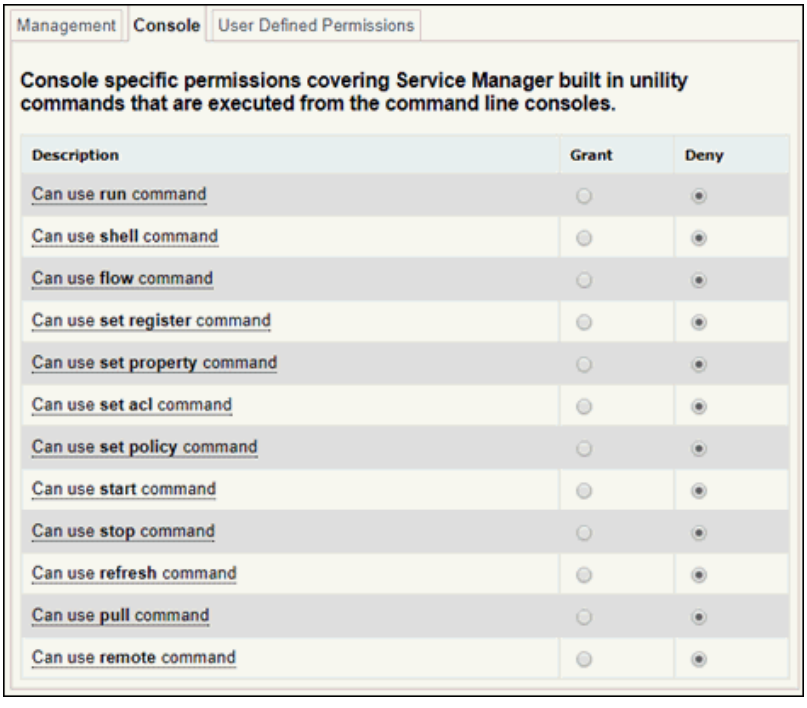
Parameter	Description
Apply To (continued)	A *regular expression* is a special sequence of characters that use a specialized syntax to develop the matching pattern. For example, if *role1* is general (blank field in *Application or Configurations*), then *role1* has the *Can use Stop Command* permission granted. Any user that has *role1* assigned, can use the console stop command on any instance of iWay Service Manager. If however the *Application or Configuration* field contains a value for *role1*, for example *base1*, then that scoping prevents the user from using the console stop command on any instance other than *base1*. If you have five Service Manager instances (for example, base, base001, baseA, base2, and baseB), you can create a regular expression patterned for the role to use. This role would have the *Application or Configuration* value of *base \d{0,3}* to limit the scope of the role to the instances with the names base, base001, and base2. The instance labeled baseA and baseB will not be included because neither baseA nor baseB match the regular expression pattern. **Note:** The Application or Configuration name of *general* is prohibited. Leave the field blank if you are creating a role with a *general* scope.

Parameter	Description
Service Manager Permissions	The Service Manager Permissions section contains permissions that grant the role holder access to functions and utilities, as well as viewing permissions defined by the administrator.

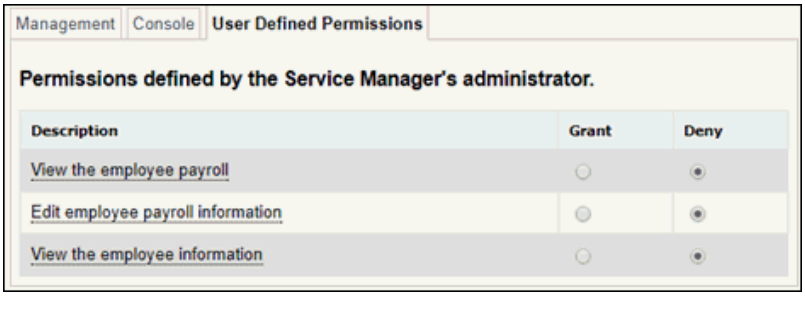
The following image shows the Management tab, that contains permissions that are used to grant the role holder access to functions that manage the iWay Service Manager instance.



The following image shows the Console tab, that contains permissions that are used to grand the role holder access to various iWay Service Manager tools.



The following image shows the User Defined Permissions tab, that show permissions that the administrator created.



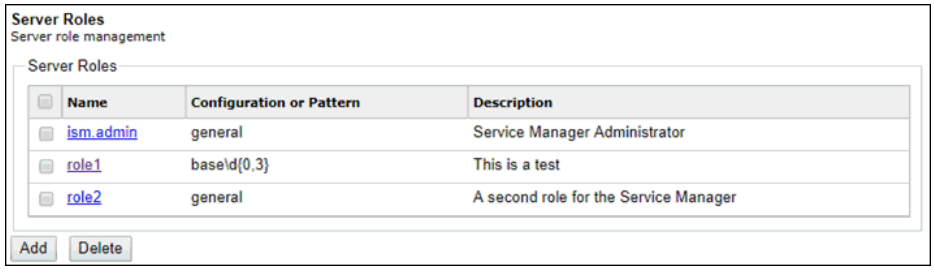
2. Once you are satisfied that the name, description, scope of the role, and permissions are correct, click *Add*.

   If you do not click the *Add* button, and instead exit the screen either by selecting another menu option or by clicking the Back button, any changes made will be lost.

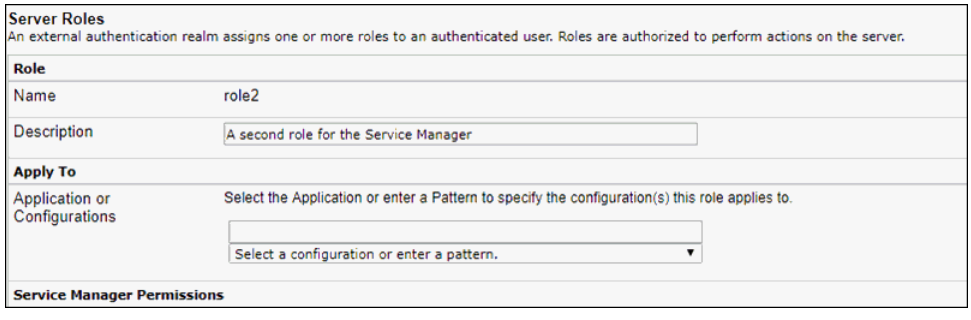
Procedure: How to Update a Server Role

To update a server role:

1. Click on the link containing the name of the role that you wish to update.

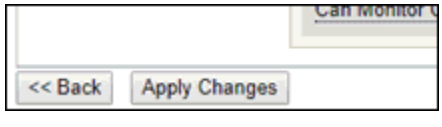


   The Service Manager Role maintenance pane appears, as shown in the following image.



   The pane is similar to the pane used to Add a Service Manager role, with the difference of the Name field, which is write-protected.

2. Update all the fields and permissions as necessary.
3. Click *Apply Changes*, as shown in the following image.

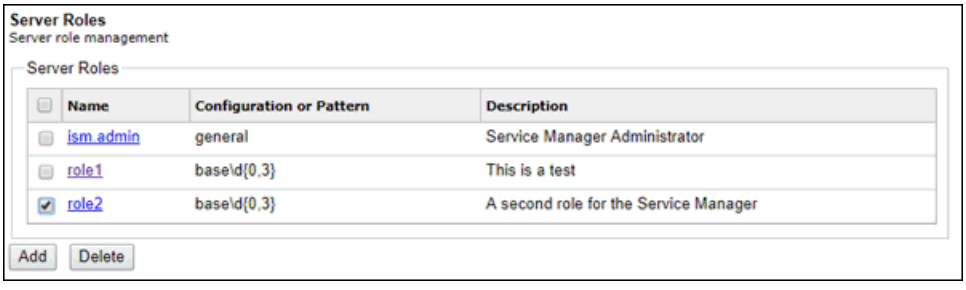


   If you do not click the *Apply Changes* button, and instead exit the screen either by selecting another menu option or by clicking the Back button, any changes made will be lost.

Procedure: How to Delete a Server Role

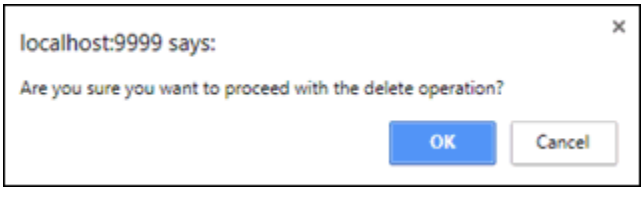
When a role is no longer needed, it can be deleted from iWay Service Manager. To delete a role from iWay Service Manager:

1. In the Server Roles list, select the check box of the role that you wish to delete, as shown in the following image.

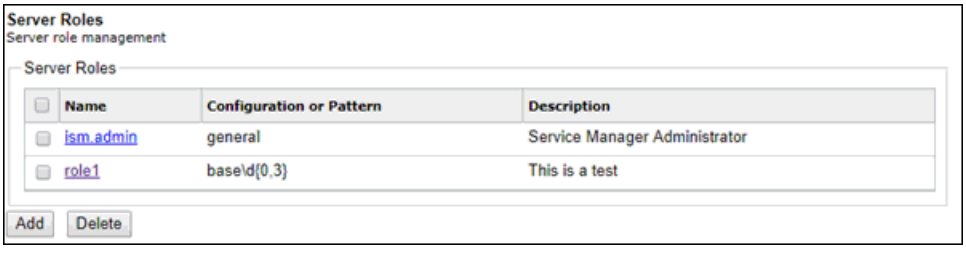


2. Click *Delete*.

   A confirmation dialog box appears, asking you if you are sure that you want to delete the operation, as shown in the following image.



3. Click *OK* to confirm deletion of the permission and have it removed from the permissions list.



   Note: Selecting the check box at the top of the list in the heading row of the table either selects all check boxes or clears them from the permission list.

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

TIBCO, the TIBCO logo, the TIBCO O logo, FOCUS, iWay, Omni-Gen, Omni-HealthData, and WebFOCUS are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.