7

Securing Web services

While you can exercise very strict control over the access to your applications from conventional 3270 terminals, your Web services clients are likely to be in remote locations, and you will be faced with new security issues.

When implementing a CICS Web services solution, you need to consider questions like the following:

- Will *authentication* be done by CICS itself or in an external server such as WebSphere Application Server?
- What *authorization* mechanisms will be used to protect access to the CICS system and access to resources such as transactions, files, and databases?
- How will you protect the *confidentiality* of data that is transported between the different tiers of the physical configuration?

In this chapter, we provide an introduction to security technologies as they relate to securing CICS Web services.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000242

7.1 Traditional CICS security

In a CICS environment, the assets you normally want to protect are the application programs and the resources that are accessed by the application programs. To prevent disclosure, destruction, or corruption of these assets, you must control access to the CICS region and to different CICS components.

You can limit the activities of a CICS user to only those functions that the user is authorized to use by implementing one or more of the following CICS security mechanisms:

- Transaction security

  This ensures that users who attempt to run a transaction are entitled to do so.

- Resource security

  This ensures that users who use CICS resources, such as files and transient data queues, are entitled to do so.

- Command security

  This ensures that users who use CICS system programming commands are entitled to do so.

- Surrogate security

  This ensures that a *surrogate* user is authorized to act on behalf of another user.

When CICS security is active, requests to attach transactions, and requests by transactions to access resources, are associated with a user ID. When a user makes such a request, CICS calls the external security manager (such as RACF) to determine if the user ID has the authority to complete the request. If the user ID does not have the correct authority, CICS denies the request.

In many cases, a user is a human operator, interacting with CICS through a terminal or a workstation. However, the user can also be a Web browser user or in a Web services solution, a program executing in a client system.

7.1.1 CICS user IDs

When a human operator signs on to a CICS region at the start of a terminal session, he or she is challenged to provide a user ID and password. The user ID remains associated with the terminal until the terminal operator signs off. Transactions executed from the terminal, and requests made by those transactions, are associated with that user ID.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000243

For connections from Web users, there are other ways that the user ID of a CICS transaction can be identified, including:

- A Web browser user is challenged to provide HTTP basic authentication information (a user ID and password). The transaction that services the client's request, and further requests made by that transaction, are associated with that user ID.
- A client program that is communicating with CICS using the Secure Sockets Layer (SSL) supplies a client certificate to identify itself. The security manager maps the certificate to a user ID. The transaction that services the client's request, and further requests made by that transaction, are associated with that user ID.

In addition to these transport-level authentication mechanisms, Web service clients can also pass authentication data in the SOAP message.

Special CICS user IDs

There are two particular user IDs that CICS uses in addition to those that identify individual end users. These are:

Region user ID The CICS region user ID is used for authorization checking when the CICS system (rather than an individual user of the system) requests access to system resources such as CICS data sets and other servers.

Default user ID When a user does not sign on, CICS assigns a default user ID to the user. It is specified in the SIT parameter DFLTUSER. In the absence of more explicit identification, it is used to identify TCP/IP clients that connect to CICS. You should give very little authority to the default user ID.

For a complete discussion of traditional CICS security, refer to *CICS TS V3.1 RACF Security Guide*, SC34-6249.

7.2 Security exposures

An end-to-end security solution addresses the security exposures found along the path of a request from an end client to a target service, including any intermediary services that route, or participate in, the service request.

To illustrate potential security exposures in a Web services environment, we use the bank teller scenario shown in Figure 7-1. The bank teller (Web service requester or client) connects over the Internet to the bank's data center where the Web service provider runs.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000244

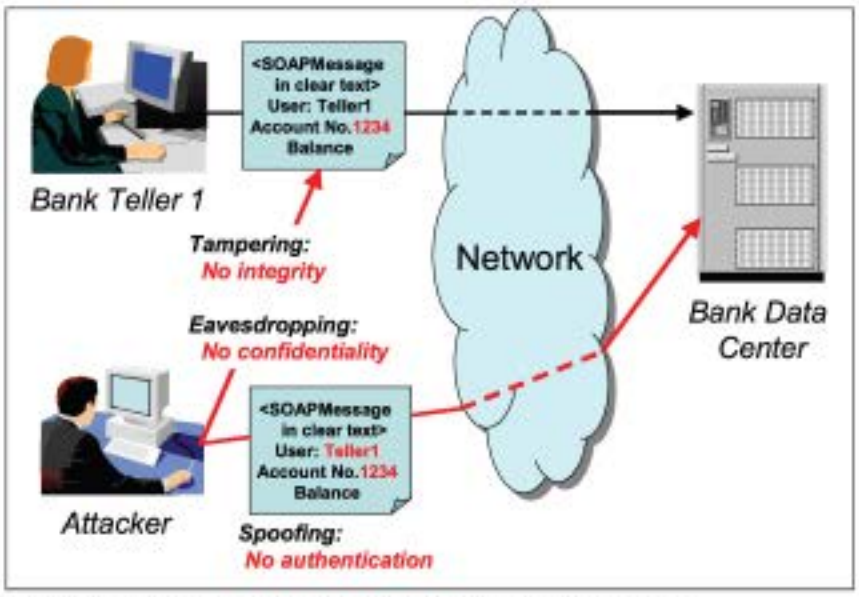


Figure 7-1 Potential security exposures in a Web services environment

If the bank has not applied any security, it has the following exposures:

- **Spoofing**

  An attacker posing as the bank teller could send a SOAP message to the service provider to get confidential information or to withdraw money from another customer's account.

  The bank can eliminate this security exposure by requiring that the bank teller authenticate herself.

- **Tampering**

  An attacker could intercept the SOAP message between the Web service requester and provider and modify the message, for example, to deposit the money into another account by changing the account number. Because there is no integrity constraint, the Web service provider does not verify whether the message has been altered and accepts the modified transaction.

  The bank can eliminate this security exposure by implementing digital signatures.

- **Eavesdropping**

  An attacker could intercept the SOAP message and read the information contained in the message because it has been sent in clear text. The attacker

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 245/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000245

could obtain confidential customer or bank information such as account numbers and balances.

The bank can eliminate this security exposure by encrypting the SOAP message.

In order to protect against the risk of these security exposures, the bank can use a combination of two types of security to secure its Web services environment:

- Transport-level security

  Transport level security mechanisms such as HTTPS can be used to secure Web services. In "Transport security" we review how different transport-level security mechanisms can be used to secure a CICS Web services solution.

- SOAP message security

  The Web Services Security model introduces a set of interrelated specifications to form a layering approach to security. When products implement these specifications, they send security information within the SOAP message itself as SOAP message headers. In "SOAP message security" on page 226 we discuss how SOAP headers can be used to secure a CICS Web services solution.

7.3 Transport security

Web services messaging relies on two protocol layers, the transport layer and the SOAP layer. Security can be implemented within either, or both, of these layers.

In this section we review how the different transport-level security mechanisms can be used to secure a CICS Web services solution. In "Comparison of transport versus SOAP message security" on page 241 we look at the main decision factors for choosing between transport-level and SOAP message security.

7.3.1 HTTP transport

When a CICS Web service is invoked using HTTP, standard HTTP security mechanisms can be used to authenticate the Web service client and to ensure message integrity and confidentiality.

Basic authentication

HTTP basic authentication is a simple challenge and response mechanism with which a server can request authentication information (a user ID and password) from a client. The client passes the authentication information to the server in an

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000246

HTTP Authorization header. The authentication information is in base-64 encoding.

The AUTHENTICATE attribute on the CICS TCPIPSERVICE resource definition specifies the authentication and identification scheme to be used for inbound TCP/IP connections for the HTTP protocol. You enable HTTP basic authentication by specifying BASIC for the AUTHENTICATE attribute.

A CICS service provider application can be protected by HTTP basic authentication. However, the HTTP basic authentication scheme can only be considered a secure means of authentication when the connection between the Web service client and the CICS region is secure. If the connection is insecure, the scheme does not provide sufficient security to prevent unauthorized users from discovering and using the authentication information for a server. If there is a possibility of a password being intercepted, basic authentication should be used in combination with SSL, so that SSL encryption is used to protect the user ID and password information.

CICS support for SSL/TLS

CICS supports both the SSL 3.0 and TLS 1.0 protocols. HTTPS connections will automatically use the TLS protocol, unless the client specifically requires SSL 3.0.

A CICS service provider application can be secured using HTTPS. HTTPS has the following advantages:

- It provides a very fast and secure transport for CICS Web services.
- It provides for authentication through either HTTP basic authentication or a client X.509 certificate.
- It provides integrity for the data passed between the service requester and CICS.
- It provides confidentiality for the data passed between the service requester and CICS by using efficient secret key cryptography.
- It can be used with hardware cryptographic devices that can significantly reduce the cost of SSL handshakes. You can customize your encryption settings to use only the cipher suites that use the Integrated Cryptographic Service Facility (ICSF).
- It is mature and similarly implemented by most vendors, and therefore, is subject to few interoperability problems.

A CICS service requester application can also use HTTPS to invoke a service provider application, with advantages similar to those listed.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000247

Enabling CICS support for SSL/TLS

To activate SSL/TLS support in a CICS TS V3.1 region, you must specify values for the following system initialization parameters:

- ENCRYPTION={STRONG | WEAK | MEDIUM}

  Specifies the cipher suites that CICS uses for secure TCP/IP connections. When a secure connection is established between the client and CICS, the most secure cipher suite supported by both is used.

  - Use ENCRYPTION=STRONG when you can tolerate the overhead of using high encryption if the other system requires it.
  - Use ENCRYPTION=WEAK when you want to use encryption keys up to 40 bits in length.
  - Use ENCRYPTION=MEDIUM when you want to use encryption keys up to 56 bits in length.

- KEYRING=*key-ring-name*

  Specifies the name of a key ring in the RACF database that contains keys and certificates used by CICS. It must be owned by the CICS region user ID. You can create an initial key ring with the DFIHSRING exec in the CICS-supplied SDFHSAMP target library.

- MAXSSLTCBS={8 | *number*}

  Specifies the maximum number of S8 TCBs that are available to CICS to process SSL connections. The S8 TCBs are created and managed in the SSL pool. An S8 TCB is only used by a task for the duration of the SSL processing.

  The maximum number of S8 TCBs that you can set is 1024. Increasing the number of available TCBs allows more simultaneous SSL connections to take place. However, increasing the number of TCBs too much will impact storage below the line.

- SSLDELAY={600 | *number*}

  Specifies the length of time in seconds for which CICS retains session IDs for secure sockets connections in the SSLCACHE. Session IDs are tokens that represent a secure connection between CICS and an SSL client. The session ID is created and exchanged between the SSL client and CICS during the SSL handshake.

  While the session ID is retained by CICS within the SSLDELAY period, CICS will re-establish an SSL connection with a client without the significant overhead of another SSL handshake. The value is a number of seconds in the range 0 through 86400. The default value is 600.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000248

Increasing the value of the SSLDELAY parameter retains the session IDs in the cache for longer, thereby optimizing the time it takes to perform SSL negotiations.

- SSLCACHE={CICS|SYSPLEX}

  Specifies whether CICS should use the local SSL cache in the CICS region, or share the cache across multiple CICS regions by using the coupling facility. If the cache is shared between a number of CICS regions, the throughput of SSL connections will improve.

  Caching across a sysplex can only take place when the regions accept SSL connections at the same IP address.

In 'Enabling SSL/TLS' on page 273 we show how we enabled our CICS region to support SSL connections from our Web service client running in WebSphere Application Server.

Activating SSL for a TCPIPSERVICE

To activate SSL support for a given connection, you must set the value of the SSL attribute on the connection's TCPIPSERVICE definition to one of the following:

- YES
  If you set the value to YES, CICS will send a server certificate to the client.
- CLIENTAUTH
  If you set the value to CLIENTAUTH, CICS will send a server certificate to the client and the client must send a client certificate to CICS.

  If you specify SSL(CLIENTAUTH) you can also specify AUTHENTICATE(CERTIFICATE) in order to map the client certificate to a RACF user ID which is then used as the identity for the client.

Cipher suites

A cipher suite must be supported by the operating system in order for CICS to use it. The list of cipher suites supported by z/OS depends on your release of the operating system. Table 7-1 and the example that follows it are for z/OS 1.4 and CICS TS V3.1. If you have a higher release of z/OS, you may have additional cipher suites available.

Table 7-1 Cipher suites

Cipher suite	Encryption algorithm	Key length	MAC algorithm
01	No encryption		MD5
02	No encryption		SHA-1

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000249

Cipher suite	Encryption algorithm	Key length	MAC algorithm
03	RC4	40 bits	MD5
04	RC4	128 bits	MD5
05	RC4	128 bits	SHA-1
06	RC2	40 bits	MD5
09	DES	56 bits	SHA-1
0A	Triple DES	168 bits	SHA-1
2F	AES	128 bits	SHA-1
35	AES	256 bits	SHA-1

The terms used in this table are:
MD5 - Message Digest algorithm
SHA-1 - Secure Hash algorithm
RC2, **RC4** - Rivest encryption
DES - Data Encryption Standard
Triple DES - DES applied three times
AES - Advanced Encryption Standard

The CICS ENCRYPTION system initialization table parameter specifies the level of encryption that CICS should use. Beginning with CICS TS V3.1, the default for the ENCRYPTION parameter is STRONG. This means that CICS can use all of the cipher suites shown in the table (and additional ones if you have a higher release of z/OS) to negotiate with clients.

Using CIPHERS to set a minimum and maximum level

You can set a minimum as well as a maximum encryption level by editing the list of cipher suites in the CIPHERS attribute on the appropriate CICS resource definition:

- URIMAP - for outbound Web service requests
- TCPIPSERVICE - for inbound HTTP

Note: You can also specify CIPHERS on the EXEC WEB OPEN command for outbound HTTP connections.

Setting a minimum encryption level provides you with more control over the security of the SSL connections that you will allow. The list of cipher suites for

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000250

each level of the ENCRYPTION parameter follows. This example shows the cipher suites available with z/OS 1.4 and CICS TS V3.1:

- WEAK - 03060102
- MEDIUM - 0903060102
- STRONG - 0504352F0A0903060102

If you do not want SSL to allow the weaker cipher suites during the SSL negotiation with the partner, then you would set a minimum level by editing the CIPHERS attribute to remove the cipher suites that you do not allow. For example, if you wanted to enforce STRONG and not allow the negotiation to drop to WEAK, you would remove the MEDIUM and WEAK cipher suites and keep the unique STRONG cipher suites. In this case your CIPHERS attribute would have 0504352F0A. If the partner does not support the stronger cipher suites, the SSL handshake will fail and the connection will not be established with the partner.

Important: If the SSL handshake negotiates down to using cipher suite 01 or 02, there is no encryption and data will be transmitted in the clear. It is recommended that you remove 01 and 02 from the list of cipher suites if even a minimum of encryption is required.

Optimizing SSL

Implementing SSL will cause an increase in CPU usage. You should only use SSL for applications that really need encrypted data flows. If you decide that you do need encryption, you can optimize the performance of SSL in your environment by:

- Utilizing the zSeries cryptographic hardware. z/OS 1.6 System SSL uses the CP Assist for Cryptographic Function (CPACF) directly. You can also customize the encryption settings to use only the cipher suites that use ICSF.
- Increasing the value of the CICS system initialization table parameter SSLDELAY so the session IDs remain in the SSLCACHE longer, which will result in only partial SSL handshakes.

  Note: The CICS system initialization table parameter SSLDELAY does not apply to caching across a sysplex.

- Increasing the value of the CICS system initialization table parameter MAXSSLTCBS so there are more S8 TCBs in the SSL pool for the SSL handshake negotiation.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 251/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000251

- Using the CICS SSLCACHE system initialization table parameter to implement SSL caching across a sysplex. If the cache is shared between a number of CICS regions, the throughput of SSL connections will improve.
- Keeping the socket open by coding SOCKETCLOSE NO on the TCPIPSERVICE definition for the PIPELINE. This is the default for HTTP 1.1 persistent sessions and removes the need to perform a full SSL handshake on the second or subsequent HTTP request.
- Only using client authentication by specifying SSL(CLIENTAUTH) on the TCPIPSERVICE definition when you really need your clients to identify themselves with a client certificate. Client authentication requires more network transmissions during the SSL handshake, and more processing by CICS to handle the received certificate.

Setting the user ID on the URIMAP

You can specify a user ID on the URIMAP to be used on behalf of a Web service client. You do this by setting the USERID attribute of the URIMAP definition. It specifies the 1 to 8 character user ID under which the Web services pipeline alias transaction is attached.

Note: If you define and install URIMAP resource definitions explicitly using CEDA, and you are using the CICS Web services assistant to create the Web services, you cannot take advantage of the dynamic installation of URIMAP resources when the PIPELINE resource is installed. This is because the user ID can also be specified on one of the parameters input to the CICS Web services assistant, in which case it would be used to define the USERID attribute of the URIMAP resource when it is created automatically during the PIPELINE scan. If you explicitly code USERID for the URIMAP, and install the URIMAP, it will override the dynamic installation of the URIMAP resource when the PIPELINE resource is installed.

A user ID that you specify in the URIMAP definition is overridden by any user ID that is obtained directly from the client.

Important: If you use a URIMAP definition to set a user ID, there is no authentication of the client's identity. You should only do this when communicating with your own client system, which has already authenticated its users and communicates with the server in a secure environment.

Order of precedence for determining the user ID when using HTTP

It is possible that for a single Web service request transported by HTTP, multiple methods for setting the user ID will be used at the same time. In this event, CICS

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000252

uses the following order of precedence for determining the user ID under which the target business logic program runs:

1. A user ID specified by a message handler, or a SOAP header processing program, that is included in the pipeline that processes the SOAP message. For example, a SOAP header processing program could extract a username from the SOAP message and specify that the CICS task should run with this user ID. See "SOAP message security" on page 226 for more information on how SOAP message security can be used with CICS and HTTP.
2. A user ID obtained from the Web client using basic authentication, or a user ID associated with a client certificate.
3. A user ID specified in the URIMAP definition for the request.
4. The CICS default user ID, if no other can be determined.

7.3.2 WebSphere MQ transport

To control security checking performed by WebSphere MQ (WMQ), you must define *switch profiles*. When a queue manager is started (or when the WMQ REFRESH SECURITY command is issued), WMQ first checks the status of RACF and the MQADMIN class. It sets the subsystem security switch off if it discovers one of these conditions:

- RACF is inactive or not installed.
- The MQADMIN class is not defined to RACF.
- The MQADMIN class has not been activated.

If both RACF and the MQADMIN class are active, WMQ checks the MQADMIN class to see whether any of the switch profiles have been defined. If subsystem security is not required, WMQ sets the internal subsystem security switch off and performs no further checks. The sequence of subsystem security checks is shown in Figure 7-2.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000253

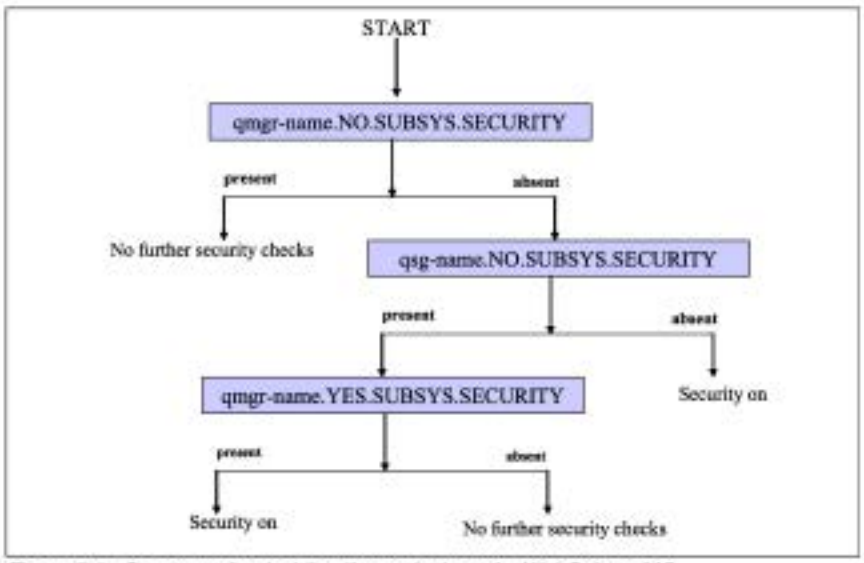


Figure 7-2 Sequence for deciding if security is on for WebSphere MQ

Switch profiles can be set at the queue manager level and at the queue-sharing group level, but the queue manager level is always checked first. If your queue manager is not a member of a queue-sharing group, then no queue-sharing group checks are made. Switch profiles are not subject to any access list checks and are merely used to indicate to WebSphere MQ whether a particular security switch is on or off. A number of switch profiles exist that can be used to control the security checking for your WebSphere MQ environment.

When using WebSphere MQ as the transport mechanism for accessing Web services in CICS, you need to consider the following points:

- The SOAP MQ inbound listener transaction (CPIL) is started by the trigger monitor using the same user ID as the trigger monitor transaction. This user ID must have UPDATE authority to the request queue and the backout queue (if this is specified).
- If AUTH=IDENTIFY is specified in the USERDATA parameter of the WMQ PROCESS definition for CPIL, then the user ID under which CPIL runs must have surrogate authority to allow it to start transactions on behalf of the user IDs in the MQ message descriptors (MQMDs) of the messages.

More information about security for WMQ can be found in *WebSphere MQ Security*, SC34-6588 and in the redbook *WebSphere MQ Security in an Enterprise Environment*, SG24-6814.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 254/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000254

SSL/TLS with WebSphere MQ

SSL/TLS can be used to secure SOAP messages that are transported using WMQ. WMQ supports Version 3.0 of the SSL protocol. You specify the cryptographic algorithms that are used by the SSL protocol by supplying a CipherSpec as part of the channel definition. WMQ also supports Version 1.0 of the Transport Layer Security (TLS) protocol.

See *WebSphere MQ Security*, SC34-6588 for more information on using SSL/TLS with WMQ.

Order of precedence for determining user ID when using WMQ

It is possible that for a single Web service request transported by WMQ, multiple methods for setting the user ID will be used at the same time. In this event, CICS uses the following order of precedence to determine the user ID under which the target business logic program runs:

1. A user ID specified by a message handler, or a SOAP header processing program, that is included in the pipeline that processes the SOAP message

   For example, a SOAP header processing program could extract a username from the SOAP message and specify that the CICS task should run with this user ID. See "Enabling SOAP message security with WMQ" on page 284 for more information on how SOAP message security can be used with CICS and WMQ.

2. A user ID obtained from the MQ message descriptor

   A message can contain message context information, such as a user ID. This information is held in the message descriptor and can be generated by the queue manager when a message is put on a queue by an application, or by the application itself. This allows the receiving application to run with the same identity as the application that put the message on the queue.

3. The CICS default user ID, if no other can be determined

Note: We found that although URIMAP resources are required for service providers using WMQ, the USERID and TRANSACTION attributes were ignored.

7.4 SOAP message security

The first version of the WS-Security specification was proposed by IBM, Microsoft, and VeriSign in April 2002. After the formalization of the April 2002 specification, the specification was transferred to the OASIS consortium:

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 255/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000255

http://www.oasis-open.org

The latest core specification, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) was standardized in March 2004.

WS-Security provides a foundational set of SOAP message extensions for building secure Web services by defining new elements to be used in the SOAP header for message-level security. It specifies the use of *security tokens, digital signatures*, and *XML encryption* to protect and authenticate SOAP messages. It specifies the use of digital signatures to provide integrity for XML elements in a SOAP message, and it specifies the use of encryption to provide confidentiality for XML elements in a SOAP message. The specification allows you to protect the body of the message or any XML elements within the body or the header. You can give different levels of protection to different elements within the SOAP message.

The advantage of using WS-Security over SSL is that it can provide *end-to-end* message-level security. This means that the message security can be protected even if the message goes through multiple services, called intermediaries. SSL security is considered to be *point-to-point*, and the data may be decrypted prior to reaching the intended recipient.

As illustrated in Figure 7-3, if the service requester identifies itself to the intermediate gateway, and the intermediate gateway identifies itself to the service provider, the target service will normally run with the identity of the intermediate gateway rather than the service requester.

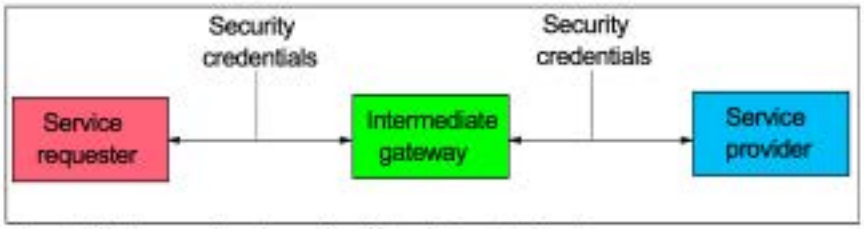


Figure 7-3 Transport-level security with an intermediate gateway

WS-Security addresses this problem by allowing security credentials to be passed within the SOAP message, so that the credentials of the service requester can be passed via an intermediate gateway, and can still be used to identify the requester to the service provider. See Figure 7-4.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 256/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000256

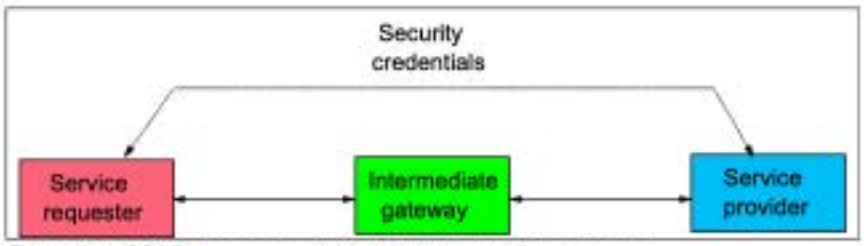


Figure 7-4 SOAP message security with an intermediate gateway

Figure 7-5 shows how a SOAP message can be extended with security data that is used to authenticate the service requester and to protect the message as it passes between the requester and the service provider. The network portion of the diagram could contain any number of intermediate nodes, some of which may not be trusted.

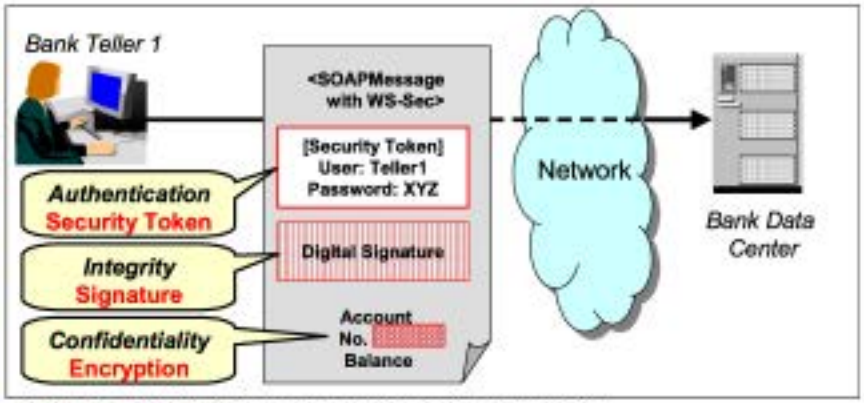


Figure 7-5 An example of a typical scenario with WS-Security

The SOAP message shown in Figure 7-5 contains three pieces of security data:

- A security token used to authenticate and identify user Teller1
- A digital signature to ensure that no one modifies the message while it is in transit without the modification being detected
- An account balance XML element that is encrypted to ensure confidentiality

To read more about the Web services security specifications, refer to:

- Specification: Web Services Security (WS-Security) Version 1.0 (April 2002):
  `http://www.ibm.com/developerworks/webservices/library/ws-secure/`

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 257/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000257

- Web Services Security Addendum (August 2002):

  http://www.ibm.com/developerworks/webservices/library/ws-secureadd.html

- Web Services Security: SOAP Message Security V1.0 (March 2004):

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

Important: When we wrote the first edition of this book, CICS did not provide support for WS-Security. However, it was still possible for a user-written SOAP header processing program to process WS-Security security tokens in CICS. We show how to do this in Chapter 8, "Security scenarios" on page 247.

7.4.1 CICS and SOAP message security

Using WS-Security, you can apply authentication, integrity, and confidentiality at the *message* level. CICS TS V3.1 provides support for WSS: SOAP Message Security through the use of a CICS-supplied message handler, DFHWSSE1, which was shipped by APAR PK22736.

The WS-Security implementation in CICS TS V3.1 contains code derived from the Apache XML Security project. The licensing terms associated with that code mean that the WS-Security implementation in CICS TS V3.1 is not licensed on the same basis as the rest of the CICS TS V3.1 product.

Except for the WS-Security implementation, CICS TS V3.1 is an "ICA Program," licensed under the relevant terms and conditions of the IBM Customer Agreement (ICA) or IBM International Customer Agreement (IICA). The WS-Security implementation is licensed under the terms and conditions of the IBM International Program License Agreement (IPLA). IPLA licensing is widely used in IBM, especially for products on distributed platforms and for one-time charge products on z/OS.

The WS-Security implementation, known as the CICS WS-Security Component, is packaged as a unique FMID with the identifier JCI640W. FMID JCI640W is licensed under the IPLA.

For this reason, the Licensed Program Specification (LPS) for CICS TS V3.1 includes a section explaining that the CICS WS-Security Component has a different licensing basis from the rest of CICS TS V3.1. Specifically, the licensing of the CICS WS-Security Component is addressed by the following three additional paper items:

- The IPLA (multi-language booklet)
- License Information (LI) document for CICS WS-Security Component (multi-language booklet)

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000258

- Proof of Entitlement for CICS WS-Security Component (multi-language sheet of paper)

Deliveries of CICS TS V3.1 after June 2, 2006 contain the additional material. Customers who ordered and received CICS TS V3.1 before that date were sent the material separately, together with the updated LPS.

Important: The WS-Security implementation in CICS Transaction Server for z/OS Version 3.1 is not licensed on the same basis as is used for the rest of the CICS TS V3.1 product.

There are several options available with the CICS WS-Security support, and which ones you choose will depend on the level of security required for the data and the transmission path of the data. The options that you can choose from are:

- Basic authentication

  In service provider mode, CICS can accept a UsernameToken in the SOAP message header for authentication on inbound SOAP messages. The UsernameToken contains a Username element and a Password element. CICS verifies the Username and Password using an external security manager such as RACF. If this is successful, CICS places the Username in container DFHWS-USERID and processes the SOAP message in the pipeline. If CICS is unable to verify the UsernameToken, it returns a SOAP fault message to the service requester.

  Username tokens are not supported on outbound SOAP messages or in service requester mode.

- Signing with X.509 certificates

  In service provider and service requester modes, you can provide an X.509 certificate in the SOAP message header to sign the body of the SOAP message for authentication. An X.509 certificate is an example of a binary security token.

  - Inbound SOAP messages

    To accept binary security tokens from inbound SOAP messages you must import the public key associated with the certificate into RACF and associate it with the keyring that is specified in the KEYRING system initialization parameter for the CICS region.

  - Outbound SOAP messages

    For outbound SOAP messages you need to generate and publish the public key to the intended recipients. The Integrated Cryptographic Service Facility (ICSF) is used to generate private keys.

- Encrypting

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 259/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000259

In service provider and service requestor modes, you can encrypt the SOAP message body using either the Triple DES algorithm or the AES algorithm. It is then included in the message and encrypted using the intended recipient's public key with the asymmetric key encryption algorithm RSA 1.5.

CICS does not support inbound SOAP messages that only have an encrypted element in the message header and no encrypted elements in the SOAP body.

- Signing and encrypting

  In service provider and service requester modes, you can choose to both sign and encrypt a SOAP message. CICS always signs the SOAP message body first and then encrypts it. This provides both message integrity and confidentiality.

Authentication

In this section, we provide an example of a SOAP message with WS-Security used for authentication.

WS-Security provides a general purpose mechanism to associate security tokens with messages for single message authentication. It does not require you to use a specific type of security token. Instead it is designed to be extensible and support multiple security token formats to accommodate a variety of authentication mechanisms. For example, a client might provide proof of identity and proof of a particular business certification.

Example 7-1 shows a sample SOAP message without applying WS-Security. The SOAP message is an Order request for our sample catalog application.

Example 7-1 SOAP message without WS-Security

```
<soapenv:Envelope
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header/>
  <soapenv:Body>
    <p635:DFHOXCHN xmlns:p635="http://www.DFHOXCHN.DFHOXCPS.Request.com">
      <p635:ca_request_id>01ORDR</p635:ca_request_id>
      <p635:ca_return_code>0</p635:ca_return_code>
      <p635:ca_response_message></p635:ca_response_message>
      <p635:ca_order_request>
        <p635:ca_userid>srthstrh</p635:ca_userid>
        <p635:ca_charge_dept>hahhhh</p635:ca_charge_dept>
        <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000260

```
        <p635:ca_quantity_req>1</p635:ca_quantity_req>
        <p635:filler1 xsi:nil="true" />
      </p635:ca_order_request>
    </p635:DFHOXCMN>
  </soapenv:Body>
</soapenv:Envelope>
```

As you can see in Example 7-1, the SOAP message does not have any SOAP headers. We will apply WS-Security by inserting a SOAP security header.

WS-Security defines a vocabulary that can be used inside the SOAP envelope. The XML element <wsse:Security> is the *container* for security-related information. (Note that wsse stands for *Web services security extension*.)

When you use WS-Security for authentication, a security token is embedded in the SOAP header and is propagated from the message sender to the intended message receiver. On the receiving side, it is the responsibility of the server security handler to authenticate the security token and to set up the caller identity for the request.

In Example 7-2 we show the same SOAP message, but this time with authentication. As you can see, we have user name and password information contained in the <UsernameToken> element.

Example 7-2 SOAP message with WS-Security

```
<soapenv:Envelope
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.
     xsd">
      <wsse:UsernameToken>
        <wsse:Username>WEBUSER</wsse:Username>
        <wsse:Password
         Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1
         .0#PasswordText">
           REDBOOKS
        </wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <p635:DFHOXCMN xmlns:p635="http://www.ibm.com/DFHOXCMN.JFHOXCPB.Request.com">
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000261

```
      <p635:ca_request_id>01ORDR</p635:ca_request_id>
      <p635:ca_return_code>0</p635:ca_return_code>
      <p635:ca_response_message></p635:ca_response_message>
      <p635:ca_order_request>
        <p635:ca_userid>srthstrh</p635:ca_userid>
        <p635:ca_charge_dept>hbhhhh</p635:ca_charge_dept>
        <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
        <p635:ca_quantity_req>1</p635:ca_quantity_req>
        <p635:filler1 xsi:nil="true" />
      </p635:ca_order_request>
    </p635:DFHOXCMN>
  </soapenv:Body>
</soapenv:Envelope>
```

The <UsernameToken> element of the SOAP message in Example 7-2 contains credentials that can be used to authenticate the user WEBUSER.

The simplest form of security token is the UsernameToken, which is used to provide a user name and password for basic authentication. In "The WS-Security header processing program" on page 267, we show an example of how a header processing program can extract a UsernameToken from a SOAP header, validate the username and password, and set the user ID of the CICS task to the username passed in the header.

A signed security token is one that is cryptographically signed by a specific authority. For example, an X.509 certificate is a signed security token.

Security token usage for WS-Security is defined in separate profiles such as the Username token profile and the X.509 token profile.

To read more about these security token standards, refer to:

- Web Services Security: UsernameToken Profile V1.0 (March 2004):

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

- Web Services Security: X.509 Token Profile V1.0 (March 2004):

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

Implementing SOAP message security in CICS

To implement WS-Security in CICS TS for either a service provider or a service requester, you must include a <wsse_handler> element in the configuration file for the appropriate pipeline. Use the following sub-elements of <wsse_handler> to provide configuration information to DFHWSSE1:

- <dfhwsse_configuration>

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000262

This element can be used in either a service provider or a service requester pipeline. It may contain the following optional elements: each of these is discussed in more detail below:

- `<authentication>`
- `<expect_signed_body/>`
- `<expect_encrypted_body/>`
- `<sign_body>`
- `<encrypt_body>`

Detailed discussion of these optional elements follows.

- `<authentication>`

  Specifies the use of security tokens in the headers of inbound and outbound SOAP messages. It can be used in either a service provider or a service requester pipeline. In a service provider pipeline, the element specifies whether CICS should use the security tokens in an inbound SOAP message to determine the user ID under which work will be processed. In a service requester pipeline, it specifies that CICS should add an X.509 certificate to the security header for outbound SOAP messages.

  The `<authentication>` element has two attributes: *trust* and *mode*. These attributes determine whether asserted identity is used and the combination of security tokens used in a SOAP message. The *trust* attribute can be set to either none, basic or signature. The *mode* attribute can also be set to either none, basic, or signature. For more information on the meaning and valid combinations of these attributes, refer to the *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SG34-6458.

  Asserted identity allows a trusted user to assert, or declare, that work should run under a different identity (the *asserted identity*), without the trusted user having the credentials associated with that identity. Messages contain a *trust token* and an *identity token*. The adjectives *trust* and *identity* indicate how the token is used rather than the kind of token. A trust token, for example, could be a UsernameToken or an X.509 token. The trust token is used to check that the sender has the correct permissions to assert identities, and the identity token holds the asserted identity (user ID) under which the request is to run.

  Note: If you use asserted identity, it requires that the service provider trusts the requester to make this assertion. In CICS, the trust relationship is established with security manager surrogate definitions: the requesting identity must have the correct authority to start work on behalf of the asserted identity.

  The `<authenticate>` element can contain the following elements:

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 263/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000263

- <certificate_label>

  Optional. Specifies the label associated with an X.509 digital certificate ignored in a service provider pipeline

- <suppress/>

  Optional. For a service provider, the handler will not use any security tokens in the message to determine under which user ID to run. For a service requester, the handler will not add to the SOAP message any of the security tokens required for authentication

- <algorithm>

  Specifies the URI of the signature algorithm. CICS currently supports the signature algorithms for inbound SOAP messages shown in Table 7-2.

Table 7-2 Signature algorithms for inbound SOAP messages

Algorithm	URI
Digital Signature Algorithm with Secure Hash Algorithm 1 (DSA with SHA1)	http://www.w3.org/2000/09/xmldsig#dsa-sha1
Rivest-Shamir-Adleman algorithm with Secure Hash Algorithm 1 (RSA with SHA1)	http://www.w3.org/2000/09/xmldsig#rsa-sha1

- <expect_signed_body/>

  Indicates that the <body> of the inbound message must be properly signed. If it is not, CICS rejects the message with a security fault.

- <expect_encrypted_body/>

  Indicates that the <body> of the inbound message must be properly encrypted. If it is not, CICS rejects the message with a security fault.

- <sign_body>

  Directs DFHWSSE1 to sign the body of outbound SOAP messages, and provides information regarding how the messages are to be signed. It can be used in either a service provider or a service requester pipeline. It contains the following elements:

  - <algorithm>

    Specifies the URI of the algorithm used to sign the body of the SOAP message.

    CICS currently supports the following signature algorithm for outbound SOAP messages:

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000264

- Rivest-Shamir-Adleman algorithm with Secure Hash Algorithm 1 (RSA with SHA1), which is specified using the URI http://www.w3.org/2000/09/xmldsig#rsa-sha1

- <certificate_label>

  Specifies the label associated with an X.509 digital certificate. The digital certificate should contain the private key since this was used to sign the message. The public key associated with the private key is then sent in the SOAP message, which allows the signature to be validated.

- <encrypt_body>

  Directs DFHWSSE1 to encrypt the body of outbound SOAP messages, and provides information regarding how the messages are to be encrypted. It can be used in both a service provider and service requester pipeline. It contains the following elements:

  - <algorithm>

    Specifies the URI identifying the algorithm used to encrypt the body of the SOAP message. CICS currently supports the encryption algorithms shown in Table 7-3.

Table 7-3 Encryption algorithms

Algorithm	URI
Triple DES in cipher block chaining mode	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
AES with a key length of 128 bits in cipher block chaining mode	http://www.w3.org/2001/04/xmlenc#aes128-cbc
AES with a key length of 192 bits in cipher block chaining mode	http://www.w3.org/2001/04/xmlenc#aes192-cbc
AES with a key length of 256 bits in cipher block chaining mode	http://www.w3.org/2001/04/xmlenc#aes256-cbc

    <certificate_label>

    Specifies the label associated with an X.509 digital certificate. The digital certificate should contain the public key of the intended recipient of the SOAP message so that it can be decrypted with the private key when the message is received.

Example 7-3 shows a <wsse_handler> element with all of the optional elements present. You would add this to your configuration file for the pipeline.

Example 7-3 <wsse_handler>

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000265

```
<wsse_handler>
 <dfhwsse configuration version="1">
   <authentication trust="signature" mode="basic">
     <certificate_label>AUTHCERT03</certificate_label>
     <suppress/>
   </authentication>
   <expect_signed_body/>
   <expect_encrypted_body/>
   <sign_body>
     <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
     <certificate_label>SIGCERT01</certificate_label>
   </sign_body>
   <encrypt_body>
     <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
     <certificate_label>ENCCERT02</certificate_label>
   </encrypt_body>
 </dfhwsse configuration>
</wsse_handler>
```

Example 7-4 shows the pipeline configuration file basicsoap11provider.xml for the EXPIPE01 service provider pipeline associated with the sample catalog application.

Example 7-4 CICS-supplied sample pipeline configuration file basicsoap11provider.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
   xmlns="http://www.ibm.com/software/htp/cics/pipeline"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Example 7-5 shows how you would modify the pipeline configuration file to add the <service_handler_list> and <wsse_handler> elements to implement WS-Security. CICS will read the pipeline configuration file and when it finds the <wsse_handler> element it will load program DFHWSSE1 from library SDFHWSLD in your DFHRPL concatenation to process the security information. For more information on the elements for the pipeline configuration file, and

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000266

which ones are contained by other elements (high-level structure diagrams), refer to the most current version of *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SG24-6458.

Example 7-5 <wsse_handler> element added to configuration file basicsoap11provider.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
    xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipelineprovider.xsd">
  <service>
    <service_handler_list>
      <wsse_handler>
        <dfhwsse configuration version="1">
          <authentication trust="signature" mode="basic">
            <certificate_label>SIGUSERID</certificate_label>
            <suppress/>
          </authentication>
          <expect_signed_body/>
          <expect_encrypted_body/>
          <sign_body>
            <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
            <certificate_label>SIGCERT01</certificate_label>
          </sign_body>
          <encrypt_body>
            <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
            <certificate_label>ENCCERT02</certificate_label>
          </encrypt_body>
        </dfhwsse configuration>
      </wsse_handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal handler>
  </service>
  <appHandler>DFHPITP</appHandler>
</provider_pipeline>
```

If CICS is the service provider, CICS will decrypt any inbound encrypted SOAP message automatically when it processes the message, provided you have the <wsse_handler> element in the pipeline configuration file. The security header in the received message provides all of the information needed for CICS to decrypt it. In other words, the <encrypt_body> and <sign_body> elements do not need to be specified in the provider pipeline configuration file in order to decrypt the

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000267

inbound SOAP message. But you can (and probably will want to) include the <encrypt> <body> or <sign> <body>, or both, in the provider pipeline configuration file if you want to encrypt or sign the reply body sent back to the requester. This is what we have shown in Example 7-5.

7.4.2 WebSphere and SOAP message security

WebSphere Application Server V6.1 is based on the implementation of WS-Security in the following OASIS specification and profiles:

- WS-I Basic Security Profile

  http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html

- Web Services Security: SOAP Message Security 1.0 (March 2004):

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

- Web Services Security: UsernameToken Profile 1.0 (March 2004).

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

- Web Services Security: X.509 Certificate Token Profile V1.0 (March 2004)

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

WebSphere Application Server uses the J2EE 1.4 Web services deployment model to implement WS-Security. The Web services security constraints are specified in the IBM extension of the Web services deployment descriptors and bindings. The Web services security runtime enforces the security constraints specified in the deployment descriptors. One of the advantages of this deployment model is that you can define the Web services security requirements outside of the application business logic. With the separation of roles, the application developer can focus on the business logic, and the security expert can specify the security requirement.

Figure 7-6 shows the high-level architecture model that is used to secure Web services in WebSphere Application Server.

Document Id: [illegible] Page 268(5)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000268

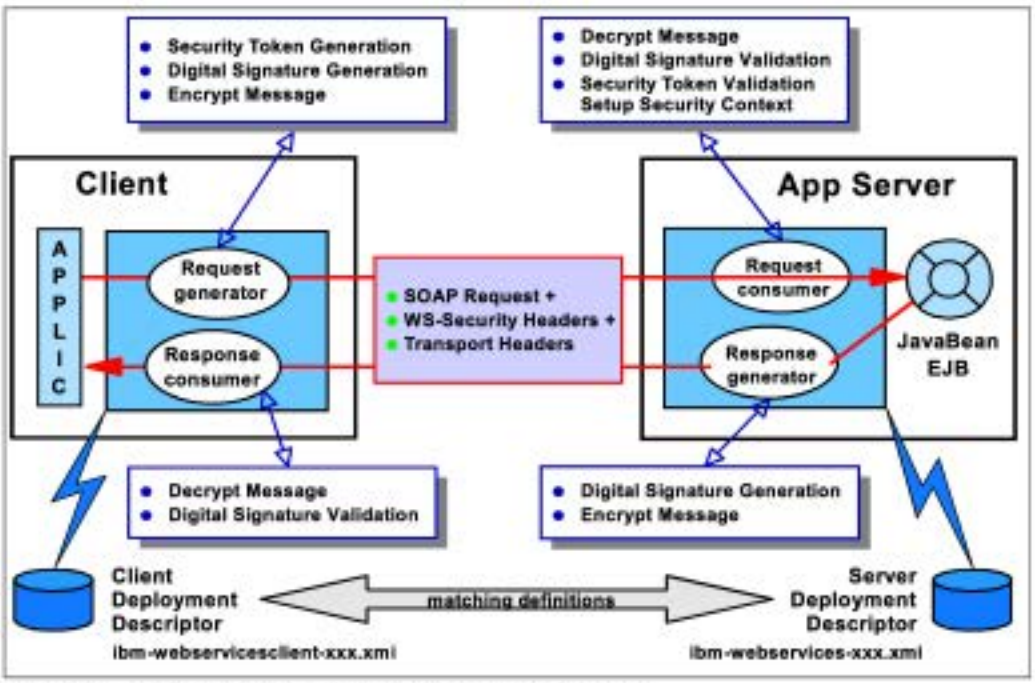


Figure 7-6 WebSphere Application Server support for WS-Security

As shown in the figure, there are two sets of configurations on both the client side and the server side.

- **Request generator**

  This client-side configuration defines the Web services security requirements for the outgoing SOAP message request. These requirements might involve generating a SOAP message request that uses a digital signature, incorporates encryption, and attaches a security token.

- **Request consumer**

  This server-side configuration defines the Web services security requirements for the incoming SOAP message request. These requirements might involve verifying that the required integrity parts are digitally signed, verifying the digital signature, verifying that the required confidential parts were encrypted by the request generator, decrypting the required confidential parts, validating the security token, and verifying that the security context is set up with the appropriate identity.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 269/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000269

- **Response generator**

  This server-side configuration defines the Web services security requirements for the outgoing SOAP message response. Those requirements might involve generating the SOAP message response with Web services security, including digital signature, and encrypting and attaching a security token, if necessary.

- **Response consumer**

  This client-side configuration defines the Web services security requirements for the incoming SOAP response. The requirements might involve verifying that the integrity parts are signed and the signature is verified; verifying that the required confidential parts are encrypted and that the parts are decrypted; and validating the security token, if necessary.

The Web services security requirements that are defined in the request generator must match the request consumer. The requirements that are defined in the response generator must match the response consumer. Otherwise, the request or response is rejected because the Web services security constraints cannot be met by the request consumer and response consumer.

The format of the Web services security deployment descriptors and bindings are IBM proprietary. However, the following tools are available to edit the deployment descriptors and bindings.

- Rational Application Developer V6.0 (RAD)

  In Chapter 8, "Security scenarios" on page 247 we show how we configured WS-Security for our sample catalog manager application using RAD.

- Application Server Toolkit V6.1 (AST)

  In Chapter 9, "Security scenarios using CICS WS-Security support" on page 295 we show how we configured WS-Security using the AST.

7.5 Comparison of transport versus SOAP message security

We have shown in this chapter that it is possible to implement Web services security at two levels: the transport level and the SOAP message level. If your Web services environment is simple (for example, it does not span multiple nodes), a security solution based on transport-level security alone may be all that you need. For more complex scenarios, however, it may not be enough on its own.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000270

In this section we provide general guidelines to help you decide what type of security solution to implement.

- You might chose to use only transport-level security to secure your CICS Web services environment when:
  - No intermediaries are used in the Web service environment or, if there are intermediaries, then you can guarantee that once the data is decrypted, it cannot be accessed by an untrusted node or process

    The transport is only based on HTTP.
  - The Web services client is a stand-alone Java program

    WS-Security can only be applied to clients that run in a Web services environment that supports the WS-Security specification (for example, WebSphere Application Server).
- You might chose to use WS-Security (possibly in addition to transport-level security) when:
  - Intermediaries are used, some of which may be untrusted.

    Security credentials that flow in the SOAP message can pass through any number of intermediaries. Protecting confidential information in the actual SOAP message may avoid the overhead of encrypting and decrypting via SSL at every intermediary node.

    Furthermore, an intermediary may be able to provide an authentication service to CICS, such that the intermediary server authenticates the Web service client and then flows an *asserted* identity to CICS.
  - Multiple transport protocols are used.

    WS-Security works across multiple transports and is independent of the underlying transport protocol.
  - The Web service partners support WS-Security and a general decision has been taken to flow security tokens in accordance with the WS-Security specification.
  - You choose to implement your own security procedures and processing by writing a custom message handler program that can process secure SOAP messages in the pipeline.

7.6 Securing CICS Web services using the service integration bus

WebSphere Application Server provides the ability to use the *service integration bus* as an intermediary between service requesters and service providers.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000271

allowing you to control the flow, routing, and transformation of messages through mediations and JAX-RPC handlers (see Chapter 5, "Connecting CICS to the service integration bus" on page 129).

The bus provides a flexible way to expose and call services located in an intranet from the Internet (and vice versa), while also providing mechanisms for protocol switching and security, including extensive support for the WS-Security specification.

Figure 7-7 illustrates how the bus could be used to enable service requesters to access CICS Web services via a *gateway service*. See "Creating a gateway service on the bus" on page 139 for information on how we enabled the inquireSingle Web service of the CICS sample catalog application to be invoked via a gateway service.

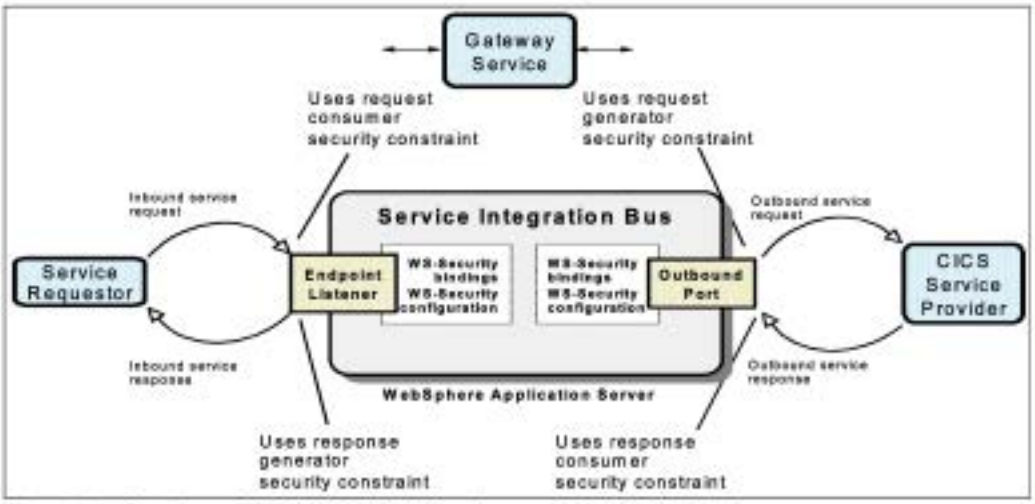


Figure 7-7 Securing a CICS Web service using the service integration bus

One advantage of such a configuration is that you can configure the bus for secure transmission of SOAP messages using tokens, digital signatures, and encryption, in accordance with the WS-Security specification. For each service, you can select the security settings that are applied between the service requester and the gateway service, and between the gateway service and the service provider (in this case, CICS).

WS-Security security constraints can be configured for the following:

- Request consumer - used on inbound requests from a client to a service.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 272/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000272

- Request generator - used when generating outbound requests from a service to a target Web service.
- Response consumer - used on outbound responses from a target Web service to a service.
- Response generator - used when generating inbound responses from a service to a client.

The bus can also invoke Web services that include https:// in their addresses. This provides message integrity and confidentiality while the message is transmitted between the bus and the CICS region.

7.7 Identity assertion

Identity assertion is an authentication mechanism that is applied among three parties: a client, an intermediary server, and a target server.

- A request message is sent to an intermediary server with a client's security token.
- The intermediary server authenticates the client and transfers the client's request message and identity to the target server with the *intermediary's* security token.

Identity assertion is an extended security mechanism supported by WebSphere Application Server Version 6.0 and the service integration bus. There are several options for sending the client's identity with the intermediary's token to the target server.

It is possible to use identity assertion to secure a CICS Web service via the bus. For example, for the Gateway service shown in Figure 7-7, we could:

- Define a security requirement for the request consumer such that the requester must provide a binary security token, such as an X.509 certificate.
- Configure the bus (WebSphere Application Server) to map the certificate to a user identity.
- Define a security requirement for the request generator such that the bus propagates the service requester's identity to CICS in a UsernameToken.

The bus must establish a trust relationship with the CICS region by authenticating itself and then by being recognized as a trusted partner of the CICS region. This can be done using one of two different modes:

Trust token	The bus sends a trust token to CICS.
Presumed trust	Trust is established at the transport level rather than at the SOAP message level.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000273

7.7.1 Trust token model

In this model, SOAP messages that flow between the bus and CICS contain a trust token and an identity token. The trust token is used to check that the sender has the correct permissions to assert identities, and the identity token holds the asserted identity (user ID) under which the request is to run. When using the CICS-provided support for WS-Security, a trust token can be a UsernameToken or an X.509 token.

The advantage of the trust token model is that it is independent of the mechanism used to transport the SOAP messages. The disadvantage, however, is the overhead of validating the trust token for each SOAP request.

7.7.2 Presumed trust model

In this model, SOAP messages that flow between the bus and CICS contain only an identity token. The trust relationship between the bus and CICS must be established using a transport-based mechanism such as SSL client authentication.

The CICS-provided support for WS-Security does not support the presumed trust model. To implement such a model, you need to write a custom handler. The user-written header processing program CIWSSECS described in "Configuring the service provider" on page 288 is an example of such a custom handler.

The advantage of the presumed trust model is that the trust established between the bus and CICS can be persistent (for example, by using SSL persistent connections) and does not need to re-established for each SOAP message. The disadvantage, however, is that it is dependent on the mechanism used to transport the SOAP messages.

Recommendation: The presumed trust model offers significant performance advantages over the trust token model.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000274

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000275

8

Security scenarios

In this chapter we outline several security scenarios that demonstrate how you can secure a CICS Web services environment. We start with an explanation of how we prepared the system and the settings that we used for the basic security configuration of our CICS system. We then provide step-by-step security configuration for a number of scenarios, including:

- Setting the user ID on a URIMAP resource definition
- Enabling SOAP message security with HTTP
- Enabling SSL/TLS
- Enabling SOAP message security with WebSphere MQ

Important: The SOAP message security scenarios documented in this chapter were tested on a version of CICS TS V3.1 which did not have the CICS WS-Security PTFs installed. See Chapter 9, "Security scenarios using CICS WS-Security support" on page 295 for scenarios using the CICS supplied WS-Security support.

We show how we configured both CICS and WebSphere Application Server for these security scenarios.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net

Page 276/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000276

8.1 Preparation

Security techniques for securing Web services were discussed in Chapter 7, "Securing Web services" on page 213. In this chapter, we describe the following security scenarios:

- Setting the user ID on a URIMAP definition

  The user ID is then used for Web service requests that match the URIMAP. No authentication of the client's credentials takes place. See "Setting the user ID on a URIMAP definition" on page 252 for details of this scenario.

- Enabling SOAP message security with HTTP

  In this scenario, we show how a header processing program in a service provider pipeline can extract a WS-Security UsernameToken from a SOAP header, validate the username and password, and set the user ID of the CICS task to the username passed in the header. This scenario is described in "Enabling SOAP message security with HTTP" on page 257.

- Enabling SSL/TLS

  In this scenario, we show how SSL/TLS can be used in combination with SOAP message security to secure a connection between WebSphere Application Server and CICS. See "Enabling SSL/TLS" on page 273.

- Enabling SOAP message security with WebSphere MQ (WMQ)

  In this scenario, we show how a header processing program in a service requester pipeline can attach a WS-Security header to a SOAP message, which is then sent, using WMQ, to a CICS service provider. This scenario is described in "Enabling SOAP message security with WMQ" on page 284.

8.1.1 Software checklist

The software we used is listed in Table 8-1.

Table 8-1 Software used in the security scenarios

Windows	z/OS
Internet Explorer V6.0	z/OS V1.6
Windows 2000 SP4	CICS Transaction Server V3.1*
IBM WebSphere Application Server - ND V6.0.2.0	RACF V1.6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000277

Windows	z/OS
Our J2EE applications ▸ CatalogSec.ear Catalog manager service requester application with no security enabled ▸ CatalogSec_WS-Security.ear Catalog manager service requester application with WS-security enabled ▸ CatalogSec_WS-Security_HTTPS.ear Catalog manager service requester application with WS-security and HTTPS enabled	Our user-written CICS programs ▸ CIWSMSGH (message handler program) ▸ CIWSSECH (header processing program) ▸ CIWSSECR (header processing program) ▸ CIWSSECS (header processing program) ▸ MYPARSER (COBOL XML parsing program)

a. The version of CICS TS V3.1 used in these security scenarios does not include the CICS support for WS-Security which was introduced later in PTFs UK15271 and UK15261.

8.1.2 Definition checklist

The definitions we used are listed in Table 8-2.

Table 8-2 Settings used in the security scenarios

Value	CICS TS	WebSphere Application Server
IP name	mvsg3.mop.ibm.com	cam21-pc3.mop.ibm.com
IP address	9.100.193.167	9.120.198.171
TCPIP port	14321	9080
SSL/TLS port	14323	
Jobname	CIWSS3C1	
APPLID	ASPOS3C1	
TCPIPSERVICE	S3C1	
Provider PIPELINE	PIPE1	
Configuration files	ITSO.7PX6.soapprovider.xml ITSO.7PX6.soaprequester.xml ITSO.7200.soapprovider.disabled.xml	
MQ queue manager	MQS3	

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000278

Value	CICS TS	WebSphere Application Server
URIMAP	SECPORDR SECICATA SECISING	

The user IDs we used in our configuration are listed in Table 8-3.

Table 8-3 User IDs

Value	CICS TS
CICS region user ID	CIWS3D
CICS default user ID	CICSUSER
User IDs to which we wish to permit access	CIWSNW WEBAS3

Tip: For the examples described in this book, we permit access to single user IDs. In a production environment, you will probably create a group of users requiring common access. Once a group is built, you can permit access to the group. This allows users' access to be controlled by the group to which they belong, rather than by individual permissions. This simplifies the security definitions required.

8.2 Basic security configuration

First we discuss our basic security configuration, taking a CICS region with no security and configuring it to enable transaction security (we do not implement other types of CICS security such as resource security and command security). We document the system initialization table parameters necessary to set up basic CICS security and then we test our basic security configuration.

For detailed information on CICS security, see *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454.

Because the INQC and INQS transactions are browse-only transactions, we chose not to secure them. The ORDR transaction updates the database, so we decided to protect it from unauthorized use.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 279/9

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000279

Our starting point is a CICS region with no security configured. This means that all transactions run under the CICS default user ID, as shown in Figure 3-12 on page 93.

8.2.1 Setting up basic security configuration

We configured our CICS region with security prefixing, transaction security, and surrogate user security active using the following SIT parameters:

- SEC=YES
- SECPRFX=YES
- XTRAN=YES
- XUSER=YES

SEC=YES was specified to indicate that we wanted RACF services to control access to CICS resources.

We used security prefixing (SECPRFX=YES) in our CICS region, which prevents our RACF security profiles from affecting other CICS regions. This is useful in a production environment since it means all security profiles are unique to an individual region, however, it can mean more work for the security administrator because more profiles must be defined.

XTRAN=YES was specified so CICS would control who could start transactions and XUSER=YES specifies that CICS is to perform surrogate user checking.

8.2.2 Testing the basic security configuration

After enabling security in our CICS region, we attempted to place an order with the sample catalog application. The Web browser received an error message indicating that the use of some transaction is forbidden (Figure 8-1).

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000280

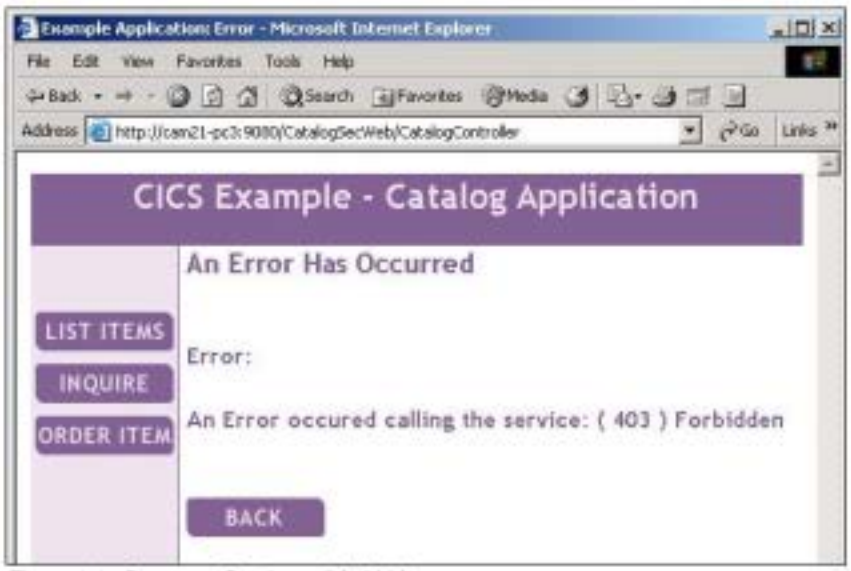


Figure 8-1 Browser - Service call forbidden

Example 8-1 shows the security violation highlighting that the CICS default user ID does not have access to the CPIH transaction.

Example 8-1 CICSUSER security violation messages running CPIH

```
DFHXS1111 11/24/2005 12:33:04 A6P0S3C1 CPIH Security violation by user CICSUSER for resource
          CIWS3D.CPIH in class TCICSTRN. SAF codes are (X'00000008',X'00000000'). ESM codes
          are (X'00000008',X'00000000').
DFHMB0361 11/24/2005 12:33:04 A6P0S3C1 An attempt to attach a CICS Web alias transaction for
          userid CICSUSER has failed because the user is not authorized to execute transaction
          CPIH. Host IP address: 9.100.193.167. Client IP address: 9.100.199.171.
          TCPIPSERVICE: S3C1.
DFHAC2003 11/24/2005 12:33:04 A6P0S3C1 Security violation has been detected term id = ????,
          trans id = CPIH, userid = CICSUSER.
```

8.3 Setting the user ID on a URIMAP definition

With this option the security identity used for the transaction is based on a statically defined user ID. We achieved this preset identification by using a URIMAP resource definition in which we specified a valid user ID (CIWSNW) in the USERID parameter. This is shown in Figure 8-2.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 281/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000281

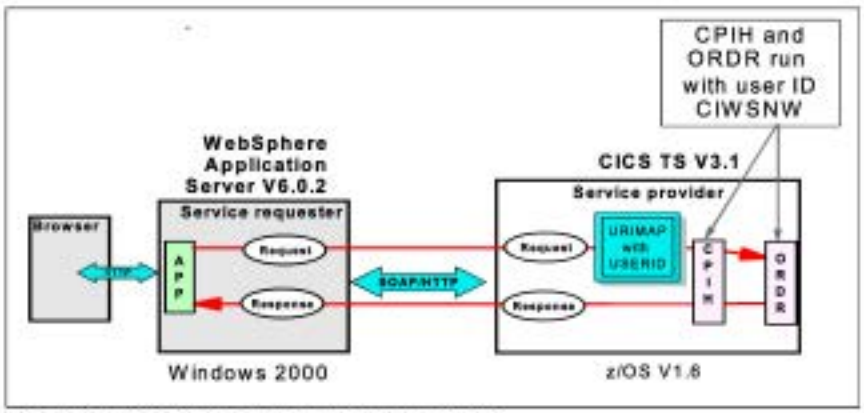


Figure 8-2 Setting user ID on a URIMAP scenario

At this stage, the only security check is to validate whether or not the user ID specified in the URIMAP attribute (CIWSNW) has permission to execute both the CPIH and ORDR transactions.

8.3.1 Defining the URIMAP

We used the following command to define a URIMAP resource:

```
CEDA DEFINE URIMAP(SECPORDR) GROUP(S3C1)
```

We then set the attributes as shown in Figure 8-3 and Figure 8-4.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000282

```
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0640
   CEDA  Define Urimap( SECPORDR )
    Urimap          : SECPORDR
    Group           : SJCI
    Description     ===>
    STatus          ===> Enabled           Enabled | Disabled
    USAge           ===> Pipeline          Server | Client | Pipeline
   UNIVERSAL RESOURCE IDENTIFIER
    SCheme          --> HTTP               HTTP | HTTPS
    HOST            --> *
    (Lower Case)    -->
    PAth            --> /exampleApp/placeOrder
    (Mixed Case)    -->
                    ===>
                    ===>
 ASSOCIATED CICS RESOURCES
    TCpipservice    ===>
 +  ANalyzer        --> No                 No | Yes
                                                      SYSID=SJCI APPLID=A6P0SJCI
```

Figure 8-3 CEDA DEFINE URIMAP (page 1 of 2)

```
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0640
   CEDA  Define Urimap( SECPORDR )
 +  COnverter     ===>
    TRansaction   ===> CPIH
    PRogram       -->
    PIpeline      --> PIPE1
    WebService    --> placeOrder                                  (Mixed Case)
   SECURITY ATTRIBUTES
    USErid        --> CICSUSR
    Ciphers       -->
    CErtificate   ===>                                            (Mixed Case)
   STATIC DOCUMENT PROPERTIES
    MediaType     ===>
    (Lower Case)
    CHaracterset  -->                                             (Mixed Case)
    HOSTCodepage  -->
    TEmplatename  -->
    (Mixed Case)
 +  H-sfile        :
                                                      SYSID=SJCI APPLID=A6P0SJCI
```

Figure 8-4 CEDA DEFINE URIMAP (page 2 of 2)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000283

We used the following values:

- URIMAP name was set to SECPORDR
- USAGE was set to PIPELINE, indicating that it is to be used for incoming Web service requests.
- SCHEME was set to HTTP because no transport security is being used.
- HOST attribute was set to the wildcard (*), meaning that the URIMAP definition matches on any host name
- PATH was set as /exampleApp/placeOrder. Together with the HOST and SCHEME values this means that the URIMAP resource caters to any Web service request originating from http://*/exampleApp/placeOrder.
- TRANSACTION is automatically defaulted to CPIH by CICS when USAGE(PIPELINE) is specified.
- PIPELINE was set to PIPE1, the name of the pipeline for which this URIMAP definition is being made.
- WEBSERVICE can only be specified when USAGE(PIPELINE) is in effect and specifies the name of the Web service to be invoked. We set the value to placeOrder
- USERID was set with the value of CICSNW, the user ID under which we have decided to execute the ORDR transaction.
- All other values were allowed to default

Two other URIMAP resources were defined with similar definitions. First SECPORDR was copied as SECICATA and as SECISING, then these resource definitions were altered as shown in Example 8-2.

Example 8-2 Creating URIMAP definitions for SECICATA and SECISING

```
CEDA COPY URIMAP(SECPORDR) GROUP(SSC1) AS(SECICATA)
CEDA COPY URIMAP(SECPORDR) GROUP(SSC1) AS(SECISING)
CEDA ALTER URIMAP(SECICATA) GROUP(SSC1) PATH(/exampleApp/inquireCatalog)
       WEBSERVICE(inquireCatalog)
CEDA ALTER URIMAP(SECISING) GROUP(SSC1) PATH(/exampleApp/inquireSingle)
       WEBSERVICE(inquireSingle)
```

This ensures that the transactions INQC, INQS, and ORDR all execute with the same user ID CICSNW.

Example 8-3 shows the RACF commands that we used to permit the CICS region user ID (CIWS3D) to start transactions (such as ORDR) with the user ID CIWSNW.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000284

Example 8-3 RACF commands to allow CIWS3D to act as surrogate for CIWSNW

```
RDEFINE SURROGAT CICSUSER.DFHSTART UACC(NONE) OWNER(CICSUSER)
PERMIT CICSUSER.DFHSTART CLASS(SURROGAT) ID(CICS3D) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

For more information regarding surrogate user security, see *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454.

8.3.2 Permitting access to user ID CICSNW

In "Writing the message handler program" on page 80 we explained how the message handler program CIWSMSGH is used to change the default transaction ID (CPIH) to a transaction ID based on the Web service request in the DFHWS-WEBSERVICE container. Example 8-4 shows the RACF commands that we used to limit access to the ORDR transaction to a single user ID (CIWSNW).

Example 8-4 RACF commands to allow access to CPIH and ORDR transactions

```
RDEFINE GCICSTRN CICS3D ADDMEM(CICS3D.CPIH) UACC(NONE) OWNER(IBMUSER)
RALTER  GCICSTRN CICS3D ADDMEM(CICS3D.ORDR)
PERMIT CICS3D CLASS(GCICSTRN) ID(CICSWNW) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

In Example 8-4 we:

- Define the transaction group CIWS3D and add the CPIH and ORDR transactions to the group
- Permit access to the group for user ID CIWSNW
- Refresh the RACF CICS transaction class TCICSTRN

8.3.3 Testing user ID on URIMAP resource definition

With these definitions installed, both the CPIH and ORDR transactions now execute with the user ID as supplied in the URIMAP. This is shown in Figure 8-5.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000285

```
INQUIRE TASK
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tas(0000050) Tra(CEMT) Fac(G312) Sus Ter Pri( 255 )
    Sta(TO) Use(CICSAA  ) Dsn(BD=61FA84DB3006)
 Tas(0000052) Tra(CPTH)           Sus Tas Pri( 001 )
    Sta(L ) Use(CICSHW  ) Lsn(BD=617FF2220DF0) Hty(G2CFAN01)
 Tas(0000053) Tra(ORDR)           Sus Tas Pri( 001 )
    Sta(L ) Use(CICSHW  ) Lsn(BD=617FF553C4774) Hty(BD=     )
 Tas(0000055) Tra(CEUF) Fac(E342) Sus Ter Pri( 001 )
    Sta(SD) Use(CICSAA  ) Dsn(BD-617FF7031E9B0) Hty(ZCIOWA: )
```

Figure 8-5 ORDR executing with preset user ID

8.4 Enabling SOAP message security with HTTP

In Section 7.4 "SOAP message security" on page 226, we explained how WS-Security extends the SOAP specification by defining new elements to be used in the SOAP header for message-level security. We also provided an outline of how WebSphere Application Server supports WS-Security, including support for generating security tokens that are passed in the WS-Security SOAP header. This scenario is shown in Figure 8-6 on page 258.

In this section we show how a simple UsernameToken security token, passed in a Web service call from WebSphere Application Server, can be processed in CICS by a user-written header processing program. The header processing program extracts the UsernameToken from the SOAP header, validates the username and password, and sets the user ID of the CICS task to the username passed in the header.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000286

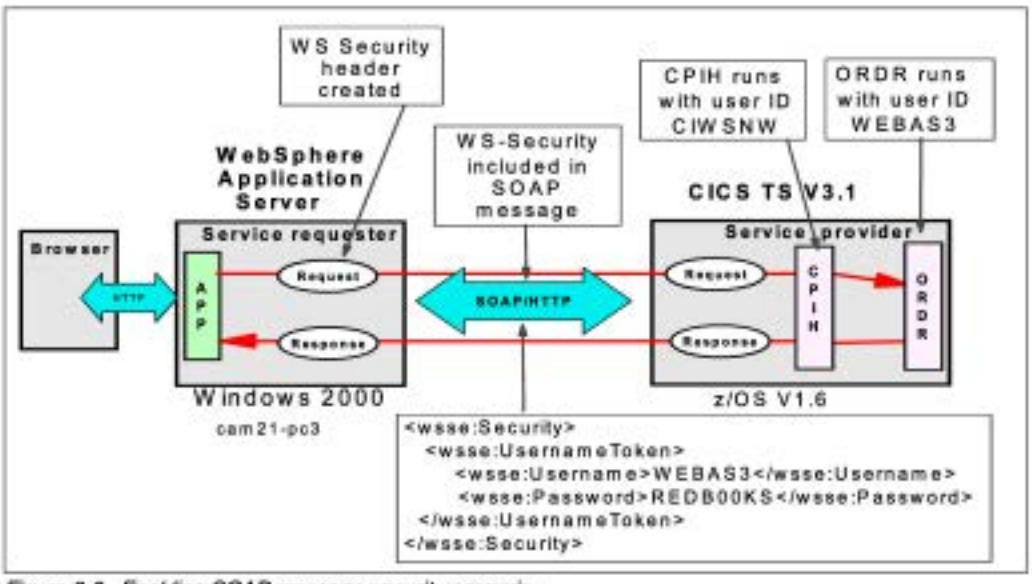


Figure 8-6 Enabling SOAP message security scenario

8.4.1 Configuring the service requester

In "Installing the service requester" on page 87 we showed how we installed our sample Web service client in WebSphere Application Server. The service requester setup tasks that we cover here are as follows:

- Defining the WebSphere WS-Security constraint for the Web service client
- Re-deploying the Web service client application

WebSphere WS-Security configuration files

The WebSphere WS-Security constraints are defined in the IBM extension of the J2EE Web services deployment descriptor. There are four configuration files: application-level deployment descriptor extensions for a client and a server, and binding files for a client and a server (Figure 8-7).

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 287/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000287

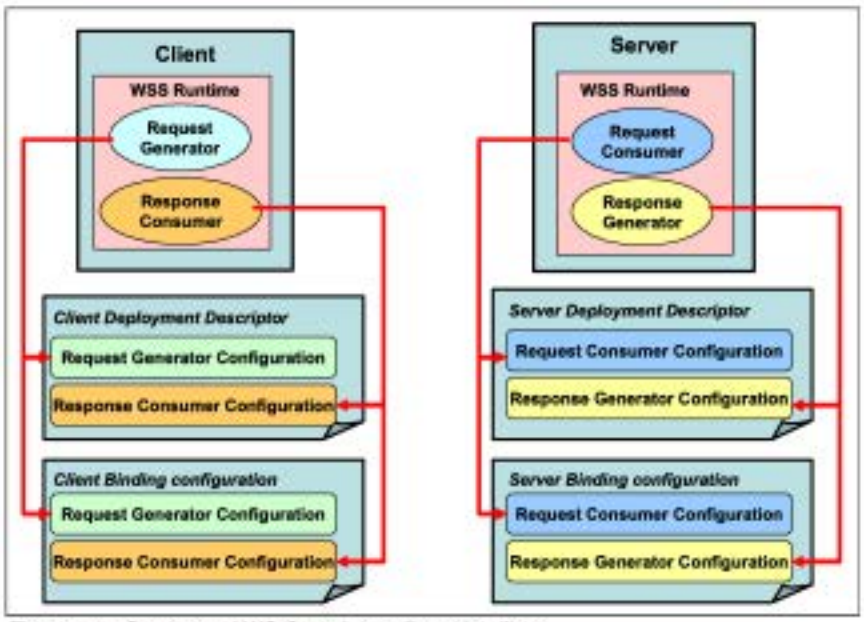


Figure 8-7 Structure of WS-Security configuration files

The configuration files are:

- Client deployment descriptor extension file - includes request generator and response consumer constraints:

  ibm-webservicesclient-ext.xmi

- Client binding configuration file - includes how to apply request generator and response consumer constraints:

  ibm-webservicesclient-bnd.xmi

- Server deployment descriptor extension file - includes request consumer and response generator constraints:

  ibm-webservices-ext.xmi

- Server binding configuration file - includes how to apply request consumer and response generator constraints:

  ibm-webservices-bnd.xmi

Note: The *.xmi* in the file names stands for *XML metadata interchange*.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineVictory.net
Page 288/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000288

The deployment descriptor extension files specify *what* security constraints are required, for example, what type of security token and whether to sign the message. The binding files specify *how* to apply the required security constraints defined in the deployment descriptor extension, for example, which security token is inserted and which keys are used for signing.

These deployment descriptor extension and binding files define the application-level security constraints and they belong to each application.

For our scenario, we needed to configure the client deployment descriptor and the client binding only, since our service provider is running in CICS and not in WebSphere Application Server.

Editing the client configuration

To configure our Web service client to use WS-Security, we used Rational Application Developer V6.0 (RAD).

Note: You can also use the Application Server Toolkit to configure WS-Security for WebSphere applications.

We imported the client application archive CatalogSec.ear into RAD. We then expanded the Dynamic Web Project (CatalogSecWeb) in the Project Explorer and opened (double-clicked) the deployment descriptor.

After the Deployment Descriptor Editor opened, we selected the **WS Extension** page. The WS Extension page is used for editing the client's deployment descriptor extension file, so you can specify what security is required. Figure 8-8 shows the WS Extension page in the Deployment Descriptor Editor.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net

Page 289/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000289

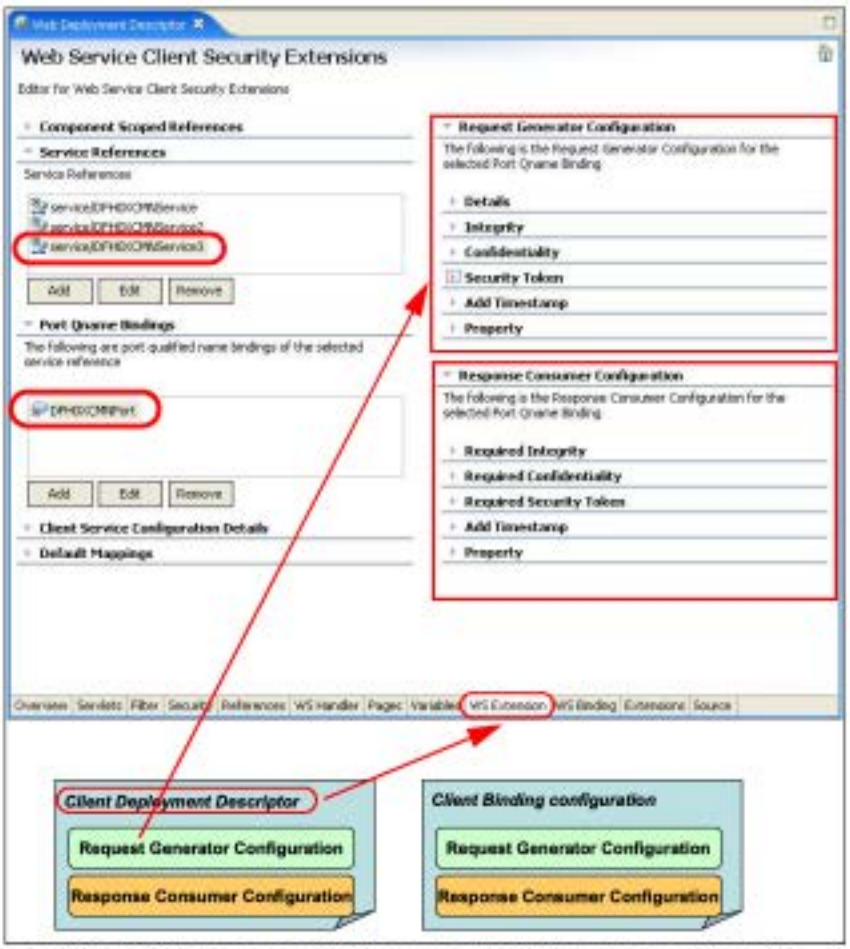


Figure 8-8 WS Extension page in Web service client Deployment Descriptor Editor

- We clicked the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPort** (the port binding for the service reference).
- In the Request Generator Configuration, we expanded **Security Token** and clicked **Add**. We entered the name `basicauth`.

  We selected the **Username** Token type from the drop-down list. When you select a Token type, the Local name is filled in automatically (Figure 8-9 on page 262). We left the URI empty.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 290/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000290



Figure 8-9 Security Token Dialog for specifying basic authentication

We clicked **OK** and a security token was created. We then saved the configuration.

After specifying the security token, a corresponding token generator must be specified in the binding configuration. The WS Binding page is for editing the client's binding file, so you can specify how to apply the required security. We clicked the **WS Binding** tab. Figure 8-10 shows the WS Binding page in the Deployment Descriptor Editor.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000291

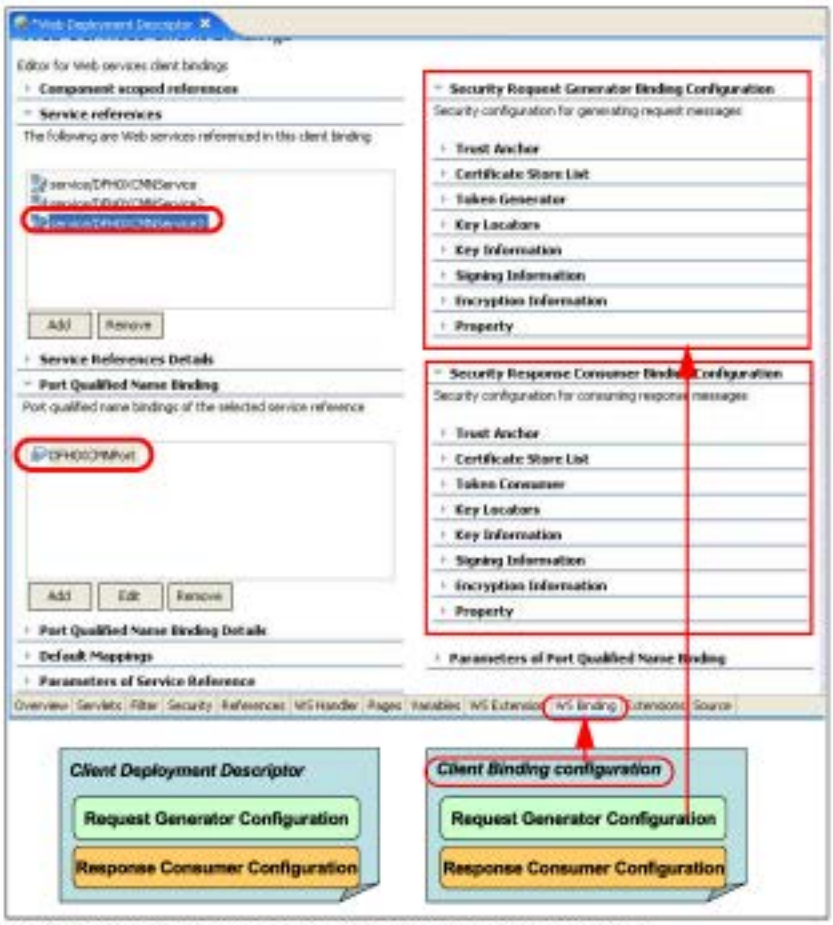


Figure 8-10 WS Binding page in the Deployment Descriptor Editor

- We selected **Token Generator** and clicked **Add**. We entered a Token generator name `basicauthToken` and allowed the Token generator class to default as `com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator`
- To select the security token, we expanded the drop-down list and selected the **basicauth** security token that we defined previously on the WS Extension page.
- We checked **Use value type** and selected the **Username Token** value type from the drop-down list. This selection filled the local name and callback handler.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 282/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000292

- We entered the user ID WEBAS3 and password REDBOOKS that we wanted to use to identify the Web service client (Figure 8-11).

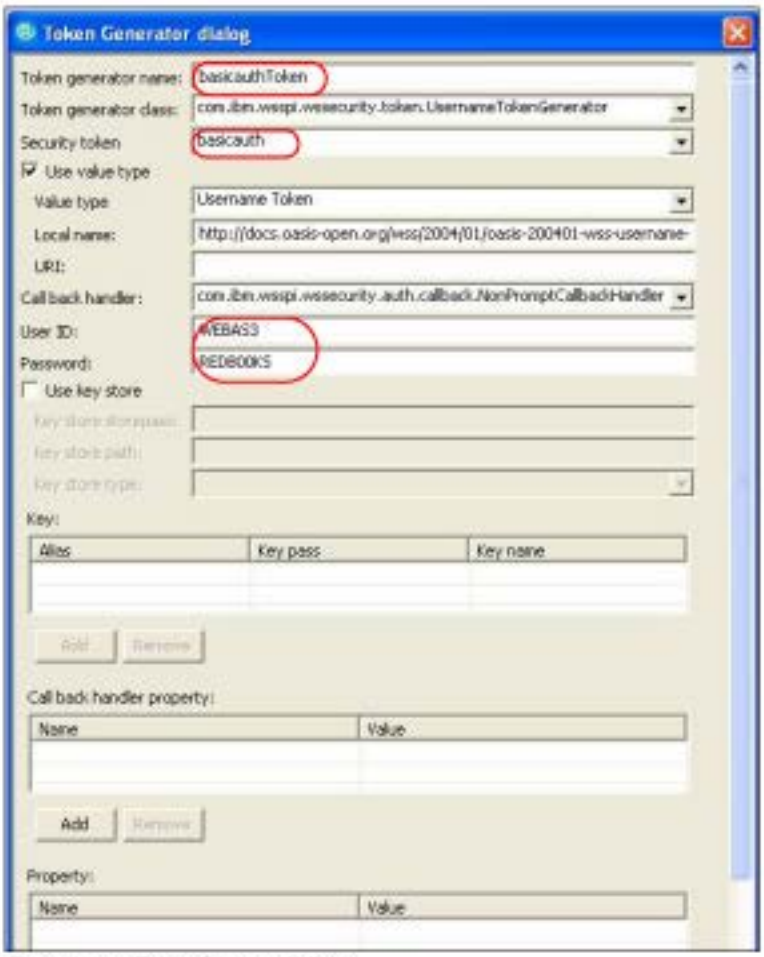


Figure 8-11 Token Generator dialog

- We clicked **OK**, and the token generator was created. We then saved the configuration.

Note: The User ID and Password in Figure 8-11 represent the credentials of the application server, *not* the browser user. The same credentials are passed to CICS in the SOAP message irrespective of the identity of the end user.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000293

Redeploying the Web service application

After configuring a security constraint for the service requester application, we exported a new EAR file called CatalogSec_WS-Security.ear. We followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec_WS-Security.ear.

8.4.2 Configuring CICS

The credentials configured using Rational Application Developer are used to form the WS-Security header that is contained in the SOAP message. An example of this message is shown in Example 8-5.

Example 8-5 SOAP security header extracted from a SOAP message

```
<wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">
    <wsse:UsernameToken>
      <wsse:Username>WEBAS3</wsse:Username>
      <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
       wss-username-token-profile-1.0#PasswordText">REDBOOK5</wsse:Password>
    </wsse:UsernameToken>
</wsse:Security>
```

This header includes the `mustUnderstand="1"` attribute, which indicates that either this header must be processed or a SOAP fault thrown.

Note: See *CICS Transaction Server for z/OS CICS Web Services Guide*, SC34-6458, for details on `mustUnderstand` and other SOAP-defined header block attributes.

In order for CICS to process the security token in the SOAP header, the pipeline configuration file must be updated to include a SOAP header processing program. Example 8-6 shows the pipeline configuration file that activates our sample security header processing program CIWSSECH.

Example 8-6 Pipeline config file, ITSO_7206_secprovider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd ">
  <transport>
    <default_transport_handler_list>
      <handler>
        <program>CIWSMSGH</program>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000294

```
        <handler_parameter_list/>
      </handler>
    </default_transport_handler_list>
  </transport>
  <service>
    <terminal_handler>
      <cics_soap_1.2_parser>
        <headerprogram>
          <program_name>CIWSSECH</program_name>
          <namespace>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
                    1.0.xsd</namespace>
          <localname>Security</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.2_parser>
    </terminal_handler>
  </service>
  <appendline>CHAFLI</appendline>
</provider_pipeline>
```

The <headerprogram> element shown in Example 8-6 introduces four new elements, <program_name>, <namespace>, <localname> and <mandatory>:

- The <program_name> element contains the name of the header processing program CIWSSECH, which is to be invoked to process the security token.
- The <namespace> element is used with the <localname> element to determine which header blocks in a SOAP message should be processed by the header processing program.
- A <localname> element is also used to determine which header blocks in a SOAP message should be processed by the header processing program. The <localname> contains the element name of the header block, in our case, `Security` (see Example 8-5 on page 265).
- The <mandatory> element indicates that the header processing program must be invoked at least once, regardless of whether or not such a header has been found.

  <mandatory>true</mandatory> has the following meaning:

  During request processing in a service provider pipeline, and response processing in a service requester pipeline, the header processing program is to be invoked at least once, even if none of the headers in the SOAP message matches the <namespace> and <localname> elements:

  - If none of the headers matches, the header processing program is invoked once.
  - If any of the headers match, the header processing program is invoked once for each matching header.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000295

During request processing in a service requester pipeline, and response processing in a service provider pipeline, the header processing program is to be invoked at least once, even though the SOAP message that CICS creates has no headers initially. If you want to add headers to your message, you must ensure that at least one header processing program is invoked, by specifying <mandatory>true</mandatory> or <mandatory>1</mandatory>.

Important: If a Web service request that contains the attribute mustUnderstand="1" (or ="true") is received by CICS, and you have not coded a <headerprogram> element in your configuration file for this header, a SOAP fault message will be created by CICS.

Example 7-7 shows the fault message that CICS created when we had not specified a header processing program for the Security header.

Example 8-7 SOAP fault message created when no header program provided

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <SOAP-ENV:NotUnderstood qname="wsse:Security"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
     rity-secext-1.0.xsd" />
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault xmlns="">
      <faultcode>SOAP-ENV:MustUnderstand</faultcode>
      <faultstring>Header block local name 'Security' is not defined to CICS.
       Mustunderstand check failed for the header block.</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The WS-Security header processing program

The header processing program CIWSSECH is invoked at every invocation of this pipeline because of the <mandatory>true</mandatory> element, but only executes entirely if the DFHFUNCTION container holds the value RECEIVE-REQUEST. Example 8-8 gives an overview of the program.

Example 8-8 Pseudo code overview of the CIWSSECH program

```
Check if invoked for RECEIVE-REQUEST, else exit
Check for correct URI, else exit
Obtain WS-Security header from DFHHEADER container
Invoke MYPARSER to parse the header
Verify the credentials extracted by MYPARSER
Put user ID into DFHWS-USERID container
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 296/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000296

Example 8-9 shows how this is done. The full program is shown in Section A.2, "Sample header processing program - CIWSSECH" on page 505.

Note: You should *not* install CIWSSECH into a production environment in its present form. It has only been coded as a example and does not contain sufficient error checking. If you wish to use it in production, you *must* review the code and make whatever changes are required for your environment.

Example 8-9 Determining if invoked to process an inbound request message

```
       GET-DFHFUNCTION.
           EXEC CICS
              GET CONTAINER('DFHFUNCTION')
              INTO(WS-FUNC-AREA)
              FLENGTH(WS-FUNC-LEN)
              RESP(WS-RESP)
           END-EXEC.

      * Copy the input container to our storage
           IF WS-FUNC-LEN NOT = 16
              MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
              PERFORM FAULT-MESSAGE
           ELSE
              IF WS-FUNC-AREA NOT = 'RECEIVE-REQUEST '
                 EXEC CICS RETURN END-EXEC
              END-IF
           END-IF.
```

Example 8-10 shows how we determined that a specific request is for the placeOrder service.

Example 8-10 Determining the requester URI

```
           PERFORM GET-DFHWS-URI.
           IF WS-URI-AREA(1:WS-URI-LEN) NOT = '/exampleApp/placeOrder'
              EXEC CICS RETURN END-EXEC
           END-IF
```

Note: The value in the DFHWS-URI container is the same value that we specified previously for the PATH attribute of the SECPORDR URIMAP in Figure 7-3 on page 192.

After verifying that we are processing a placeOrder request, the Security header (see Example 8-5 on page 265) is parsed by calling the MYPARSER program. MYPARSER is a user-written COBOL program that uses the COBOL XML PARSE statement to parse through the Security header, extracting the user ID

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000297

and password. This extracted data is then returned in a COMMAREA for the calling program to use. Example 8-11 shows the procedure division for the MYPARSER program.

Example 8-11 MYPARSER COBOL program procedure division

```
PROCEDURE DIVISION.
      MAIN-PROCESSING SECTION.
      *    Validate the container
           PERFORM INIT-AND-VALIDATE.
      * Received a valid COMMAREA so invoke the XML parser
      * passing it the XML message
           XML PARSE XML-DOCUMENT
              PROCESSING PROCEDURE XML-HANDLER
           END-XML.
           PERFORM RETURN-RESPONSE.
           EXEC CICS RETURN END-EXEC.
      MAIN-PROCESSING-EXIT. EXIT.
```

The XML PARSE statement parses the data in XML-DOCUMENT, using the XML-HANDLER procedure. The XML-EVENT field is evaluated and if a match is found, the code for that event is executed. The START-OF-ELEMENT and CONTENT-CHARACTERS events are handled as shown in Example 8-12

Example 8-12 Extract of code from MYPARSER program

```
      XML-HANDLER SECTION.
           EVALUATE XML-EVENT
              WHEN 'START-OF-ELEMENT'
      * check if we have an element of interest
      * i.e. Username and Password
                 IF XML-TEXT = 'wsse:Username'
                    MOVE 'Y' TO IN-ELEM-USERNAME-XML-TAG-FOUND
                 ELSE
                    IF XML-TEXT = 'wsse:Password'
                       move 'Y' to in-elem-pwd
                    end-if
                 END-IF
              WHEN 'CONTENT-CHARACTERS'
      * If we are in an element we are interested in,
      * then extract its value
                 IF IN-ELEM = 'Y'
                    PERFORM EXTRACT-USER-ID
                 ELSE
                    IF IN-PWD-REQ = 'Y'
                        PERFORM EXTRACT-PASSWORD
                    END-IF
                 END-IF
              WHEN 'END-OF-ELEMENT'
```


Document: CD1W2F3 6827 11E7 B96D A6EF9A1 6682
CreationDate: XX



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000298

```
                CONTINUE
             WHEN 'START-OF-DOCUMENT'
                CONTINUE
```

Most of the other XML event types are handled with the CONTINUE statement because we have no need to process them. The full MYPARSER program is shown in Section A.4, "Sample XML parser program - MYPARSER" on page 522.

The parsed credentials are then verified, as shown in Example 8-13. If the user ID and password verify successfully, then the user ID is stored into the DFHWS-USERID container. This causes the business logic transaction ORDR to be executed with this user ID.

Example 8-13 Verifying the flowed credentials

```
             EXEC CICS VERIFY
                PASSWORD(CA-PASSWORD)
                USERID(CA-USER-ID)
                RESP(WS-RESP)
             END-EXEC
*       Succesful ?
             IF WS-RESP = DFHRESP(NORMAL)
                PERFORM SET-USER-ID
             ELSE
                MOVE WS-NOT-AUTH TO WS-FAULT-STRING
                PERFORM FAULT-MESSAGE
             END-IF
```

Note: In the same way that we are using the header processing program for validating security credentials, it is quite feasible to use such a program to provide some form of audit trail.

We completed the CICS setup for this scenario by permitting access to the ORDR transaction to the user ID WEBAS3:

```
PERMIT CIWS3D.ORDR CLASS(GCICSTRN) ID(WEBAS3) ACCESS(READ)
```

8.4.3 Testing SOAP message security

Now when we submit an order request on the Web browser, the credentials are added to the SOAP Security header and are extracted by the header processing program CIWSSECH. Figure 8-12 shows the ORDR transaction running with the user ID WEBAS3, while the CPIH transaction continues to execute with the CIWSNW user ID.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000299

```
STATUS:  RESULTS - OVERTYPE TO MODIFY
  Tas(0000139) Tra(CEMT) Fac(0135) Ter Pri( 255 )
    Sta(TO) Use(CICSA  ) Dsn(BCFAD2535FEF8767)
  Tas(0000459) Tra(CPIH)           Sus Tas Pri( 001 )
    Sta(L ) Use(CICSHN  ) Log(BDFAD5D4E2ED1740) Hty(0205400)
  Tas(0000460) Tra(ORDR)           Sus Tas Pri( 001 )
    Sta(L ) Use(WEBAS3  ) Log(BDFAD5D46F43FF1BF) Hty(BDF     )
  Tas(0000467) Tra(CIHF) Fac(0414) Sus Ter Pri( 001 )
    Sta(SD) Use(CICSA  ) Dsn(BCFAD5D4E52AE4BE1 Hty(2C20AAC0)

                                                    SrSID=SIC) APPLID=A6P053CL
```

Figure 8-12 CADR executing with SOAP security header user ID

8.4.4 SOAP fault messages

The <Fault> element in a SOAP message is used for reporting errors. CICS provides an application programming interface for creating, adding, or deleting SOAP fault messages. See the *CICS Transaction Server for z/OS CICS Application Programming Reference*, SC34-6434, and *CICS Transaction Server for z/OS CICS Web Services Guide*, SC34-6458, for further details on these commands.

If the credentials extracted do not validate, then CIWSSECH creates a SOAP fault message (Example 8-14) which is returned to the service requester.

Example 8-14 Generating a SOAP fault

```
           EXEC CICS VERIFY
              PASSWORD(CA-P455WORD)
              USERID(CA-USER-ID)
              RESP(WS-RESP)
           END-EXEC
      *    Successful ?
           IF WS-RESP = DFHRESP(NORMAL)
              PERFORM SET-USER-ID
           ELSE
              MOVE WS-NOT-AUTH TO WS-FAULT-STRING
              PERFORM FAULT-MESSAGE
           END-IF
         .
         .
         .
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000300

```
 FAULT-MESSAGE SECTION.
*---------------------------------------------------------------*
* Generate a SOAP Fault
*---------------------------------------------------------------*
     EXEC CICS
         GET CONTAINER('DFHWS-SOAPLEVEL')
         INTO(WS-SOAP-LEVEL)
         FLENGTH(WS-HEAD-LEN)
         RESP(WS-RESP)
     END-EXEC.

* MOVE CORRECT VERSION OF FAULTCODE
     IF WS-SOAP-11 MOVE DFHVALUE(CLIENT) to WS-FAULT-CODE
     ELSE
        MOVE DFHVALUE(SENDER) to WS-FAULT-CODE
     END-IF

     EXEC CICS SOAPFAULT CREATE
               FAULTSTRING(WS-FAULT-STRING)
               FAULTSTRLEN(length of WS-FAULT-STRING)
               FAULTCODE(WS-FAULT-CODE)
     END-EXEC.
 FAULT-MESSAGE-END. EXIT.
```

Figure 8-13 shows the fault string `Not authorized to place order`, which occurs when the user's credentials are invalid.



Figure 8-13 Browser - SOAP fault message for 'not authorized' error

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 301/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000301

If the user ID and password supplied in the SOAP Security header validate correctly, but the user ID has no access to the ORDR transaction, then a RACF error message is issued, as seen in Example 8-15.

Example 8-15 RACF messages for insufficient access

```
ICH408I USER(CIWSPC  ) GROUP(SYS1    ) NAME(PAOLO )
  CIWS3D.ORDR CL(TCICSTRN)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

Figure 8-14 shows the fault string that is generated by CICS for this error.



Figure 8-14 Browser - Internal server error

8.5 Enabling SSL/TLS

By default, SOAP messages are flowed in clear text. This is likely to be unacceptable in many cases. In order to address this, some form of encryption is required. SSL/TLS is a well understood and popular way of encrypting message flows. When the client connects with SSL/TLS, privacy of the data is obtained by encrypting the data.

In this section we show how we configured both the CICS and WebSphere environments to use an SSL/TLS connection (Figure 8-15).

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
Online/Notary.net
Page 302/89

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000302

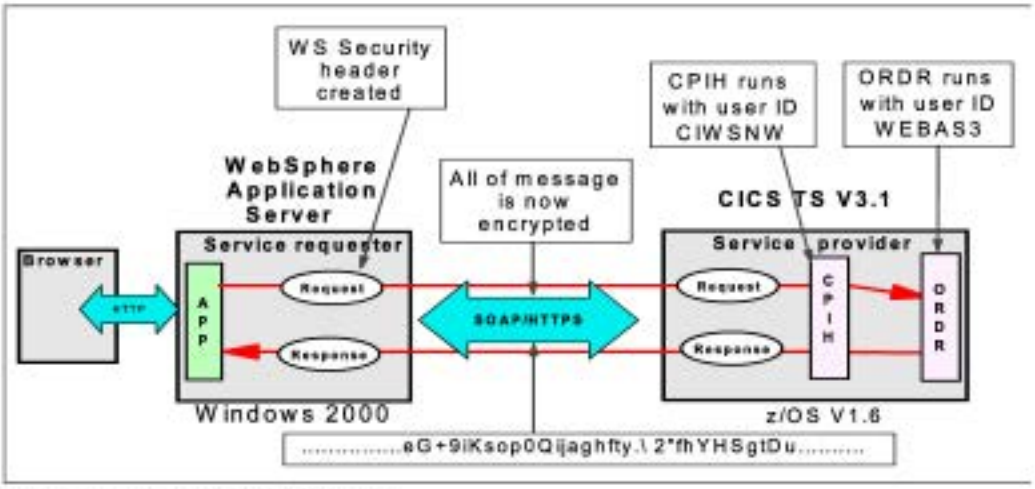


Figure 8-15 Enabling SSL/TLS scenario

We document the following steps that we followed to enable SSL/TLS:

- Creating a key ring and certificates on z/OS for CICS
- Enabling an SSL/TLS connection from WebSphere
- Configuring CICS support for SSL/TLS

8.5.1 Creating a key ring and certificates on z/OS for CICS

In order to create the certificates required for using SSL/TLS, we first needed to prepare RACF. We issued the commands shown in Example 8-16. For more information, see the chapter titled "Configuring CICS to use SSL" in *CICS Transaction Server for z/OS RACF Security Guide, SC34-6454*.

Example 8-16 RACF commands to prepare for running the DFH$RING exec

```
RDEFINE FACILITY (IRR.DIGTCERT.ADD)
RDEFINE FACILITY (IRR.DIGTCERT.CONNECT)
RDEFINE FACILITY (IRR.DIGTCERT.GENCERT)

SETROPTS RACLIST(FACILITY) REFRESH
SETROPTS GENERIC(FACILITY) REFRESH

PERMIT IRR.DIGTCERT.*       CLASS(FACILITY) ID(CIWS3D) ACCESS(READ)
PERMIT IRR.DIGTCERT.CONNECT CLASS(FACILITY) ID(CIWS3D) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CIWS3D) ACCESS(CONTROL)
```


Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineNotary.net
Page 303/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000303

The sample clist DFH$RING is provided by CICS TS to help you build a RACF KEYRING for CICS SSL use. We copied the DFH$RING clist from hlq.SDFHSAMP into our own private library and modified the clist using the customization values shown in Table 8-4.

Table 8-4 Customizing the DFH$RING clist - Site settings

Name	Value
organization	TSO
department	PSSC
certifier	CICS-Sample-Certificator
city	Montpellier
state	Herault
country	FR

These values represent the entity for which we are creating the certificates and key ring. As can be seen, these were chosen to indicate the IBM site where the Redbook residency was held.

Further parameters were supplied as input to the clist for the specific key ring to be created (Table 8-5).

Table 8-5 Customizing the DFH$RING clist - Key ring settings

Name	Value
hlqname	Cews
aslname	Cewss3c1
ipname	mvsg3.mop.ibm.com
userid	CIWS3D

The name of the key ring to be created is Cews.Cewss3c1.KeyRing. The userid parameter was supplied as CIWS3D (the CICS region user ID). The ipname is the address of the z/OS on which our CICS region CIWSS3C1 runs.

Further modifications were made to the clist because we chose not to generate certificates for a Web server or for an EJB container. Instead we generated certificates for a Web service Client, Web service Server, and a default certificate. These changes are shown in Example 8-17.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000304

Example 8-17 Customizing the DFHSTRNG EXEC - certificate labels

```
'RACDCERT ID('faruser') DELETE(LABEL('lastname' Web service Client'))'
'RACDCERT ID('faruser') DELETE(LABEL('lastname' Web service Server'))'
'RACDCERT ID('faruser') DELETE(LABEL('lastname' Default Certificate'))'
```

We used the batch job shown in Example 8-18 to execute the DFHSRING clist in order to create the required RACF profiles

Example 8-18 Batch job to invoke the DFHSRING clist

```
//RUNJOB  JOB 1,CICS,TIME=1440,NOTIFY=&SYSUID,REGION=4M,
//        CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//*
//RUN     EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 %DFHSRING CICS CICSSAC1 mvsg1.mop.ibm.com HOSTSERV(CICSSSL)
/*
```

To complete this part of the configuration we exported the Certificate Authority (CA) certificate in Base64 X.509 format using the RACF RACDCERT command:

```
RACDCERT CERTAUTH EXPORT(LABEL('CICS-Sample-Certification')) +
DSN(CICSSG4.CERTAUTH.X509) FORMAT(CERTB64)
```

8.5.2 Enabling an SSL/TLS connection from WebSphere

To enable an SSL/TLS connection from WebSphere we did the following:

- FTPed the certificate from z/OS to our client machine
- Imported the certificate into the WebSphere Application Server keystore
- Enabled the service requester to use HTTPS

FTP the certificate from z/OS

We used FTP as shown in Example 8-19 to send the exported certificate in ASCII format to our client machine.

Example 8-19 Using FTP to send an exported CA certificate to a client workstation

```
C:\>ftp mvsg3.mop.ibm.com
Connected to mvsg3.mop.ibm.com.
220-FTPD1 IBM FTP CS V1R6 at MVSG3.pssc.mop.ibm.com, 13:23:13 on 2005-11-30.
220 Connection will close if idle for more than 5 minutes.
User (mvsg3.mop.ibm.com:(none)): ciwsga
```


Document Id: CD1W2F3 6827 11E7 B96C 4CE29A1 6BEC
Confidentiality: IBM



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000305

```
331 Send password please.
Password:
230 CIWSGA is logged on.  Working directory is "CIWSGA.".
ftp> ascii
200 Representation type is Ascii NonPrint
ftp> get CERTAUTH.X509
200 Port request OK.
125 Sending data set CIWSGA.CERTAUTH.X509
250 Transfer completed successfully.
ftp: 1022 bytes received in 0.00Seconds 1022000.00Kbytes/sec.
ftp> quit
221 Quit command received. Goodbye.
```

Importing the certificate

Next, we needed to make the CERTAUTH.X509 certificate available for use by WebSphere Application Server. To do this, we imported the CERTAUTH.X509 certificate to the WebSphere Application Server keystore (DummyServerKeyFile.jks) using iKeyMan.

Note: The iKeyMan tool is provided by the IBM Java SDK. On our workstation the tool was located in directory C:\Program Files\IBM\Java142\jre\bin.

After opening a command prompt window we set the PATH variable and typed ikeyman to invoke the iKeyMan GUI.

```
set PATH=C:\Program Files\IBM\Java142\jre\bin\;%PATH%
ikeyman
```

We used iKeyMan to add the CERTAUTH.X509 file to the WebSphere Application Server keystore (ServerKeyFile.jks) as shown in Figure 8-16.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000306

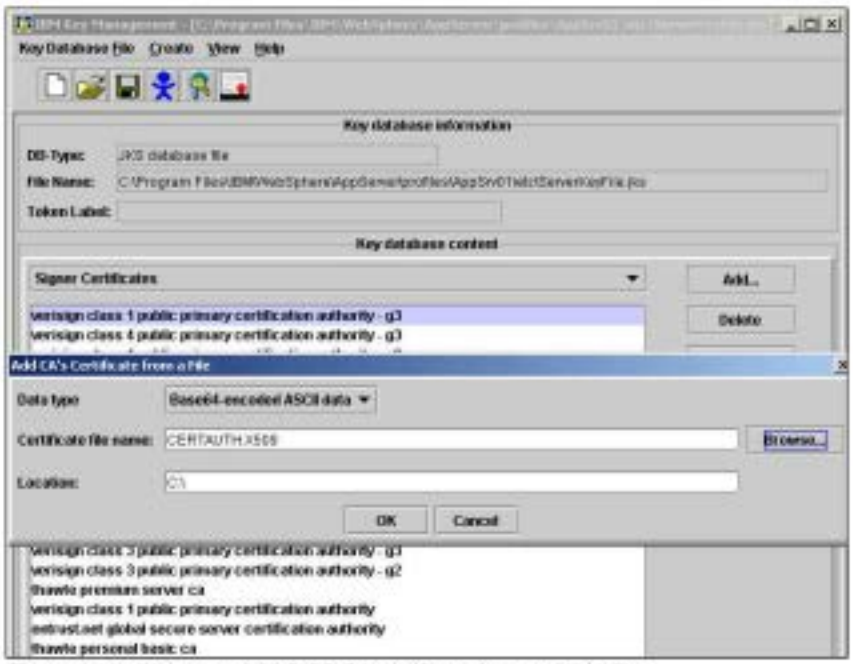


Figure 8-16 IKeyman - Add CICS CA certificate from an X509 file

After we clicked **OK** to proceed with this action, we were prompted to enter a label for this certificate. Figure 8-17 shows the CICS CA certificate added to the WebSphere keystore.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 307/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000307

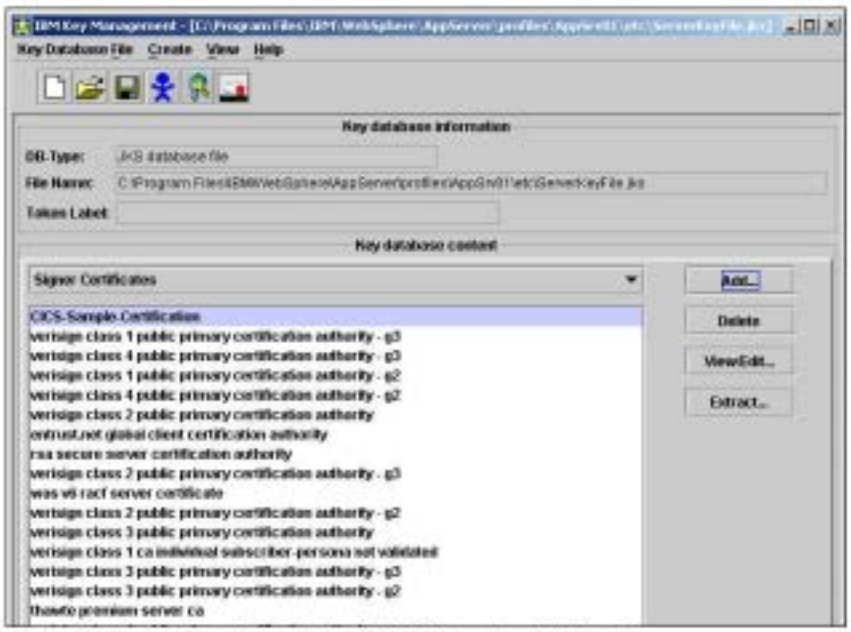


Figure 8-17 IKeyMan - After adding CICS CA certificate to keystore

Next we created a new JSSE repertoire by logging onto the admin console and clicking **Security → SSL → New JSSE repertoire**. We entered the Alias name as `CICS-SSLSettings` and defaulted the Protocol to SSL_TLS. The Key file and Trust file names were both specified as `c:\Program Files\IBM\WebSphere \Appserver\profiles\AppSrv01\bin\ServerKeyFile.jks` and the password specified was `WebAS`. Changed fields are shown in Table 8-6, all other values were allowed to default.

Table 8-6 Altered values for new JSSE repertoire

Field name	Value
Alias	CICS-SSLSettings
Key file name	C:\Program Files\IBM\WebSphere\AppServer\profiles\ AppSrv01\bin\ServerKeyFile.jks
Key file password	WebAS
Trust file name	C:\Program Files\IBM\WebSphere\AppServer\profiles\ AppSrv01\bin\ServerKeyFile.jks
Trust file password	WebAS

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 308/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000308

We clicked **OK** → **Save** → **Save** to save the configuration. Next we stopped and restarted the application server, in order to have the new repertoire recognized.

Enabling the service requester to use HTTPS

In Section 8.4.1, "Configuring the service requester" on page 258 we showed that to configure our Web service client to use WS-Security, we used Rational Application Developer V6.0. We next used a similar process to enable the service requester to use HTTPS:

1. We imported the client application archive CatalogSec_WS-Security.ear into RAD. We then expanded the Dynamic Web Project (`CatalogSecWeb`) in the Project Explorer and opened (double-clicked) the deployment descriptor.
2. After the Deployment Descriptor Editor opened, we selected the **WS Binding** page. The WS Binding page is for editing the client's binding file, so you can specify how to apply the required security.
3. We clicked the service reference **service/DFH0XCMNService** and then the Port Qualified Name Binding **DFH0XCMNPort.**
4. In the Port Qualified Name Binding Details section we specified our SSL repertoire name `Alp3-ITSO1Node01/CICS-SSLSettings` for the name of the HTTP SSL Configuration (Figure 8-18).



Figure 8-18 RAD - Enabling HTTPS for service requester

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000309

5. We repeated steps 3 and 4 for our other CICS Web services `service/DFHOXCMNService2` and **`service/DFHOXCMNService3`**.
6. We then saved the configuration and exported a new EAR file called CatalogSec_WS-Security_HTTPS.ear.

Note: The fully qualified name, including the node name, must be specified for HTTP SSL configuration: Name in order for this to be recognized by WebSphere Application Server when the application is deployed.

After enabling the service requester to use HTTPS, we needed to re-deploy the application using the WebSphere admin console. We followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec_WS-Security_HTTPS.ear.

See "Testing SSL/TLS" on page 282 for information on how we tested SSL/TLS connectivity from WebSphere Application Server to CICS.

8.5.3 Configuring CICS support for SSL/TLS

To activate SSL/TLS support in our CICS TS V3.1 region, we specified the following system initialization parameter:

- `KEYRING=Ciws.Ciwss3cl`

  This specifies the name of a key ring we created in the RACF database that contains keys and certificates used by this CICS region.

We allowed the ENCRYPTION, MAXSSLTCBS, SSLCACHE and SSLDELAY system initialization parameters to default. We then restarted the CICS region.

Tip: You should restart your CICS region after updating the system initialization parameter and before creating the TCPIPSERVICE for SSL. The CIPHERS attribute of the TCPIPSERVICE resource is only automatically updated by CICS if CICS has been started with SSL support active.

Changing the TCPIPSERVICE

To enable SSL connections in CICS we created a new TCPIPSERVICE with the PORTNUMBER, CERTIFICATE and SSL attributes set as shown in Figure 8-19.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000310

```
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0640
  CEDA  Alter TCpipservice( SSCISSL  )
   TCpipservice   : SSCISSL
   GROup          : SSCJ
   DEscription    ==> TCPIPSERVICE
   Urm            ==> DFHWBADX
   POrtnumber     ==> 14303              1-65535
   STatus         ==> Open               Open | Closed
   PROtocol       ==> Http               Iiop | Http | Eci  User
   TRansaction    ==> CWXN
   BAcklog        ==> 00005              0 27hr
   TSqprefix      ==>
   Ipaddress      ==>
   SOcketclose    ==> No                 No | 0-240000 (HHMMSS)
   MAxdatalen     ==> 000032             3-524288
  SECURITY
   SSl            ==> Yes                Yes | No | Clientauth
   CErtificate    ==> Class3el-Web-service-Server
 +  (Mixed Case)

                                                          SYSID SSCI APPLID WMPSSCI
```

Figure 8-19 TCPIPSERVICE Updated for SSL support

We noticed that CICS automatically initialized the CIPHERS attribute with a default list of acceptable codes (Example 8-20).

Example 8-20 TCPIPSERVICE CIPHERS attribute

```
CIphers      ==> 0504353E373839F3031323330A1613100009152F3F0C03060201
```

8.5.4 Testing SSL/TLS

To test the SSL/TLS scenario, we needed to configure the catalog manager J2EE application so that it accesses the catalog services using HTTPS. We started a Web browser session and entered the URL:

```
http://cam21-pc3:9080/CatalogWeb/Welcome.jsp
```

We clicked **CONFIGURE** and specified the encrypted addresses of the Catalog services (Figure 8-20). We clicked **SUBMIT** to complete the update.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000311

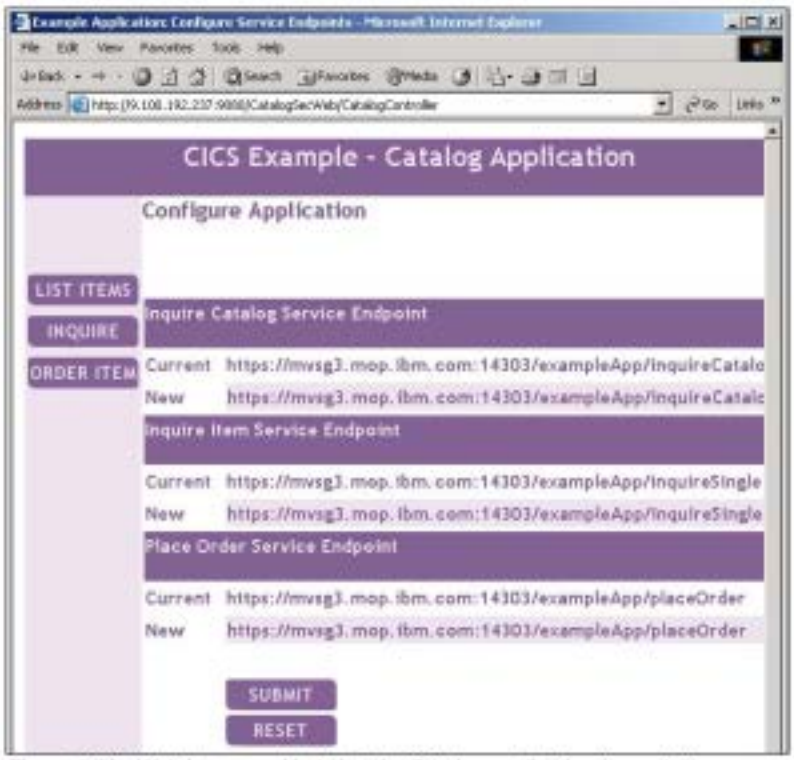


Figure 8-20 Web browser - Updating the Catalog application to use https

We then tested each of the three services (inquireCatalog, inquireSingle, and placeOrder) to ensure that all worked correctly. A display of TCP/IP connections on z/OS showed a connection to CICS region CIWSS3C1 on port 14303 (Example 8-21).

Example 8-21 Display of IP connections

```
Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF ULOG  CONSOLE CIWSGA                          LINE 42      COLUMNS 42- 121
COMMAND INPUT ===>                                             SCROLL ===> CSR
 -D TCPIP,,NETSTAT,CONN
  EZZ2500I NETSTAT CS V1R6 TCPIP 328
  USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
  BPXOINIT 00000011 0.0.0.0..10007         0.0.0.0..0             LISTEN
  CIWSR3C1 00008C23 0.0.0.0..13301         0.0.0.0..0             LISTEN
  CIWSR3C1 00008C87 9.100.193.167..13301   9.100.199.156..4338    ESTBLSH
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000312

```
CICSS3C2 00000025 0.0.0.0..13332        0.0.0.0..0          LISTEN
CICSS3D1 00000B96 9.100.193.167..14303  9.100.192.237..1086 ESTBLSH
CICSS3C1 00000B71 0.0.0.0  14333        0.0.0.0  0          LISTEN
```

A CEMT INQ TCPIPSERVICE command shows that this port has the SSL attribute and that one connection has been established (Figure 8-21).

```
 INQUIRE TCPIPSERVICE
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Tcpips(SECSSL  ) Ope Por(14302) Http Ssl Tra(CWXN)
     Con(00001) Bac( 00000 ) Max( 000032 ) Urm( CICSWEB2 ) Soc
                                                        SYSID=5EC1 APPLID=AGPCS3C1
```

Figure 8-21 TCPIPSERVICE - SSL support installed and active

8.6 Enabling SOAP message security with WMQ

In this scenario, we show how SOAP Security headers can be used with WMQ. We customize the catalog application so that when the ORDR transaction executes, it calls a Web service to dispatch the order. This Web service is located in a separate CICS region and the transport mechanism used is WMQ. To make the dispatchOrder service (CICS transaction DISP) run with the same user ID as the calling transaction (ORDR), we transfer the caller's identity in a SOAP Security header.

This scenario was chosen for two reasons.

- To demonstrate the use of a SOAP Security header in a CICS service request
- To demonstrate the use of SOAP Security headers with WMQ

Figure 8-22 shows how we configured the environment.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000313

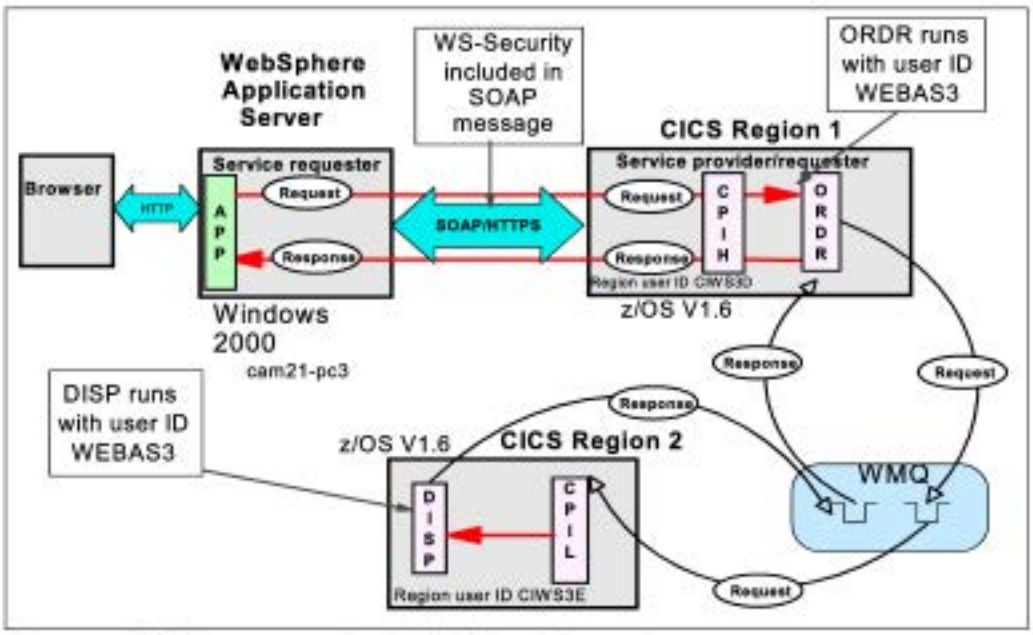


Figure 8-22 SOAP message security with WebSphere MQ scenario

Note: This scenario is an example of *identity assertion*, in which an intermediary server (CICS region 1) authenticates the client and transfers the request message and identity to the target server (CICS region 2). See "Identity assertion" on page 244 for more information on the concept of identity assertion.

We document the following steps necessary to set up this security scenario:

- Configuring CICS to use WMQ
- Configuring the service requester
- Configuring the service provider
- Configuring security for WMQ

8.6.1 Configuring CICS to use WMQ

For this, we rely heavily upon the work already done and described in Chapter 4, "Web services using WebSphere MQ" on page 111. We note only where differences occur, otherwise we used the same values.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineLibrary.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000314

We added WebSphere MQ support to our two CICS regions CIWSS3C1 and CIWSS3C2. For the dispatchOrder service, the service provider is CIWSS3C2 and the service requester is CIWSS3C1. Table 8-7 shows the different settings, parameters, and values used for these two CICS regions.

Table 8-7 Different settings for the two CICS regions

	CICS Region 1	CICS Region 2
WMQ queue manager	MQS3	MQS3
INITQ	WSB3.S3C1.INITQ	WSB3.S3C2.INITQ
Pipeline	PIPE2	PIPE3
Configuration file	/CIWS/S3C1/config/ITSO.7206.secrequester.xml	/CIWS/S3C2/config/ITSO.7206.secprovider.dispatch.xml
HFS structure	/CIWS/S3C1/...	/CIWS/S3C2/...
Header processing program	CIWSSEC1	CIWSSEC3
Region user ID	CIWS33	CIWS3E
Jobname	CIWSS3C1	CIWSS3C2
Applid	A6PCS3C1	A6PCS3C2

The WebSphere MQ queuenames used were WSB3.S3C2.PIPE3.REQUEST and WSB3.S3C2.PIPE3.RESPONSE while the process name was WSB3.S3C2.PROCESS.

The security of the CICS transactions supplied by WebSphere MQ resources was protected by issuing the RACF commands shown in Example 8-22.

Example 8-22 RACF commands to protect WebSphere MQ CICS transactions

```
RDEFINE GCICSTRN WS3MQUSR UACC(NONE) +
         ADDMEM(CIWS3E.CKBM, CIWS3E.CKCN, CIWS3E.CKDL, -
                CIWS3E.CKCP, CIWS3E.CKQC, CIWS3E.CKRE, -
                CIWS3E.CKRT, CIWS3E.CKSD, CIWS3E.CKSQ, -
                CIWS3E.CKPC, CIWS3E.CKTB, CIWS3E.CKRC, -
                CIWS3E.CKHQ, CIWS3E.CKSS, CIWS3E.CKSA) -
         OWNER(IBMUSER)

PERMIT WS3MQUSR CLASS(GCICSTRN) ID(STL)
RALTER GCICSTRN CIWS3E.ADDMEM(CIWS3E.CKTL, CIWS3E.CKAM) OWNER(IBMUSER)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

In Example 8-22 we:

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000315

- Define the transaction group WS3MQUSR and add transactions CKBM, CKCN, CKDL, CKDP, CKQC, CKRS, CKRT, CKSD, CKSQ, CKMC, CKMH, CKRC, CKRQ, CKSG and CKSV to the group.
- Permit access to the transaction group WS3MQUSR to the RACF user group SYS1 (SYS1 is the group of administration user IDs on our system)
- Alter the transaction group CIWS3T (used for CICS region CIWSS3C2) by adding transactions CKTI and CKAM to the group.
- Refresh the RACF CICS transaction class TCICSTRN.

These commands were repeated with the prefix of CIWS3D for the CIWSS3C1 CICS region

8.6.2 Configuring the service requester

In this scenario the service requester is the ORDR transaction itself. This demonstrates that a transaction can act as both a service provider and a service requester.

Configuring the sample application

We configured the catalog application in the same way as in "Configuring the Catalog application" on page 122, except that we specified the Outbound WebService URI as:

```
jms:/queue?destination=VSG3.S3C2.PIPE3.REQUEST2NQS3&targetService=/e
xampleApp/dispatchOrder&replyDestination=VSG3.S3C2.PIPE3.RESPONSE
```

This means that the SOAP request for the dispathOrder service is sent using WMQ, via the WMQ queue VSG3.S3C2.PIPE3.REQUEST, to an application servicing that queue. The response is returned to the ORDR transaction via the queue VSG3.S3C2.PIPE3.RESPONSE.

8.6.3 Header processing program

In order to add a Security header to the orderDispatch SOAP message, we wrote a new header processing program CIWSSECR that inserts a <wsse:Security> header into the message. A synopsis of this program is shown in Example 8-23.

Example 8-23 Header processing program to insert the SOAP security header

```
Check if invoked for SEND-REQUEST, else exit
Check for correct URI, else exit
Obtain user ID from DFHWS-USERID container and insert into Security header
Put Security header to DFHHEADER container
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000316

The full CIWSSECR program is shown in A.5 "Sample header processing program - CIWSSECR" on page 527.

Pipeline configuration file

A new requester pipeline configuration file was created by copying the CICS-supplied sample from basicsoap11requester.xml to /CIWS/S3C2/config/ITSO_7206_secrequester.xml. The header processing program was then added to the configuration file as shown in Example 8-24. The <program_name> element contains the name of the header processing program CIWSSECR, which is to be invoked to create the security token.

Example 8-24 Configuration file for pipeline PIPE2

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/
                    pipeline requester.xsd ">
 <service>
  <service_handler_list>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>CIWSSECR</program_name>
          <namespace>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-ws
           security-secext-1.0.xsd</namespace>
          <localname>Security</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
  </service_handler_list>
 </service>
</requester_pipeline>
```

In CICS, the new PIPELINE resource definition was created by copying PIPE1 as PIPE2 and modifying the CONFIGFILE attribute to the name of the new pipeline configuration file.

8.6.4 Configuring the service provider

A new header processing program CIWSSECS and pipeline configuration file were required.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000317

Header processing program

Program CIWSSECS is similar to CIWSSECH except that no password verification is done because no password has been sent. A short description of this program is shown in Example 8-25.

Example 8-25 Header processing program logic to change user ID

```
Check if invoked for RECEIVE <SOAP>, else exit
Check for correct URI, else exit
Obtain Security header from DFHHEADER container
Parse the header and obtain user ID
Put user ID into DFHWS-USERID container
```

Pipeline configuration file

The new configuration file /CIWS/S3C2/config/ITSO_7206_secprovider_c spatch.xml, was created by copying /CIWS/S3C1/config/ITSO_7206_secprovider.xml (see Example 8-6 on page 265). The C WSSECS program was specified in place of CIWSSECH.

Example 8-26 Configuration file for pipeline PIPE3

```
<?xml version="1.0" encoding="UTF-8"?>
<provider pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <transport>
      <default_transport_handler_list>
          <handler>
             <program>CIWSMSG</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
  </transport>
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>CIWSSECS</program_name>
          <namespace>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-ws
          security-secext-1.0.xsd</namespace>
          <localname>Security</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
```


Document is CONFIDENTIAL per IBM Agreement
Do not copy

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000318

```
  </service>
  <appendler>DFHPITP</appHandler>
</provider pipeline>
```

8.6.5 Configuring WebSphere MQ for security

Since in this scenario we are passing a user ID from CICS region 1 to CICS region 2 without re-authenticating, it is necessary to establish a trust relationship between the CICS regions. We can do this by restricting access to the specific WMQ queues used to transport the dispatchOrder request.

We secured the WMQ connection between CICS region 1 and CICS region 2 as follows:

- Configure general security for WMQ

  The RACF commands that we issued in order to configure general security for WMQ are listed in Example 8-27.

  Example 8-27 RACF commands to configure WebSphere MQ security

```
SETROPTS CLASSACT(MQADMIN,MQQUEUE,MQPROC,MQNLIST,MQCONN,MQCMDS)
SETROPTS  GENERIC(MQADMIN,MQQUEUE,MQPROC,MQNLIST,MQCONN,MQCMDS)
RDEFINE MQADMIN MQS3.NO.CMD.CHECKS
RDEFINE MQADMIN MQS3.NO.CMD.RESC.CHECKS
RDEFINE MQADMIN MQS3.NO.PROCESS.CHECKS
RDEFINE MQADMIN MQS3.NO.NLIST.CHECKS
RDEFINE MQADMIN MQS3.NO.CONTEXT.CHECKS
RDEFINE MQADMIN MQS3.NO.ALTERNATE.USER.CHECKS
RDEFINE MQADMIN MQS3.YES.SUBSYS.SECURITY
RDEFINE MQCONN  MQS3.CICS UACC(NONE)
PERMIT MQS3.CICS CLASS(MQCONN) ID(CICSSC,CICS3E) ACCESS(READ)
SETROPTS RACLIST(MQQUEUE) REFRESH
SETROPTS GENERIC(MQQUEUE) REFRESH
SETROPTS RACLIST(MQCONN) REFRESH
SETROPTS GENERIC(MQCONN) REFRESH
SETROPTS RACLIST(MQADMIN) REFRESH
SETROPTS GENERIC(MQADMIN) REFRESH
```

  The SETROPTS CLASSACT and SETROPTS GENERIC commands activate the specific and generic WMQ security classes. All classes were activated.

- Turn security on for the queue manager.

  We set the switch cmgr-name YES SUBSYS SECURITY to *Y* for our queue manager MQS3 (see "WebSphere MQ transport" on page 224).

Document ID: SG24-6827-01 ITSO Draft as of 2004-06-07
Confidential: No



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000319

- Grant access to the WMQ queues

  We granted both CICS region user IDs connect access to the queue manager and then refreshed the RACF profiles. Once those definitions were active, we then secured access to the queues, using the commands shown in Example 8-28.

Example 8-28 RACF commands to secure WebSphere MQ queues

```
RDEFINE GMQQUEUE ORDR.DISPATCH UACC(NONE) +
       ADDMEM('MQS3.VSG3.S3C2.PIPE3.REQUEST, +
              MQS3.VSG3.S3C2.PIPE3.RESPONSE')
PERMIT ORDR.DISPATCH CLASS(GMQQUEUE) ID(CICS3D,CICS3E) ACCESS(UPDATE)
RDEFINE GMQQUEUE S3C2.INITQ UACC(NONE) +
       ADDMEM('MQS3.VSG3.S3C2.INITQ')
RDEFINE GMQQUEUE S3C2.INITQ UACC(NONE) +
       ADDMEM('MQS3.VSG3.S3C2.INITQ')
PERMIT S3C2.INITQ    CLASS(GMQQUEUE) ID(CICS3D) ACCESS(UPDATE)
PERMIT S3C2.INITQ    CLASS(GMQQUEUE) ID(CICS3E) ACCESS(UPDATE)
PERMIT ORDR.DISPATCH CLASS(GMQQUEUE) ID(CICS3D) ACCESS(UPDATE)
PERMIT ORDR.DISPATCH CLASS(GMQQUEUE) ID(CICS3E) ACCESS(UPDATE)
PERMIT ORDR.DISPATCH CLASS(GMQQUEUE) ID(WEBAS3) ACCESS(UPDATE)
```

In Example 8-28 we add the response and request queues to the generic group ORDR.DISPATCH. By default, when CICS makes an API resource security check on a CICS connection, it checks to see if two user IDs have access to the resource. The first user ID checked is the CICS region user ID. The second user ID checked is the user ID associated with the CICS transaction. The CICS region user IDs (CICS3D and CICS3E) are permitted UPDATE access to the group. The user ID WEBAS3 is also granted UPDATE access.

For full details on securing access to WMQ queues, see *WebSphere MQ for z/OS System Setup Guide V5.3*, SC34-6583.

8.6.6 Testing security with WMQ

To test the WMQ security scenario, we started a Web browser session and entered the URL:

```
http://car21-pc3:9080/CatalogWeb/Welcome.jsp
```

Figure 8-23 shows the result of the initial ORDER ITEM request.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000320



Figure 8-23 Browser - INVREQ calling dispatch server

We noticed the following RACF message in the CICS log of region CIWSS3C2 (Example 8-29).

Example 8-29 RACF message when WEBAS3 has no access to DISP

```
ICH408I USER(WEBAS3  ) GROUP(CIWS    ) NAME(WEB USER1
   CIWS3E.DISP CL(TCICSTRN)
   INSUFFICIENT ACCESS AUTHORITY
   ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

This error shows that user ID WEBAS does not have access to the DISP transaction. We permitted access to the DISP transaction to the user ID WEBAS3 using the RACF commands shown in Example 8-30.

Example 8-30 RACF commands to define DISP and grant access to it

```
RDEFINE GCICSTRN WS3DISP ADDMEM(CIWS3E.DISP) UACC(NONE) OWNER(IBMUSER)
PERMIT WS3DISP CLASS(GCICSTRN) ID(WEBAS3) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
RDEFINE SURROGAT WEBAS3.DFHSTART UACC(NONE) OWNER(WEBAS3)
PERMIT WEBAS3.DFHSTART CLASS(SURROGAT) ID(CIWS3E) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

We then repeated the ORDER ITEM request successfully. Figure 8-24 on page 293 shows the DISP transaction executing with the user ID of WEBAS3 on CIWSS3C2, with CPIQ and CPIL executing with the user ID of CIWS3E.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 321/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000321

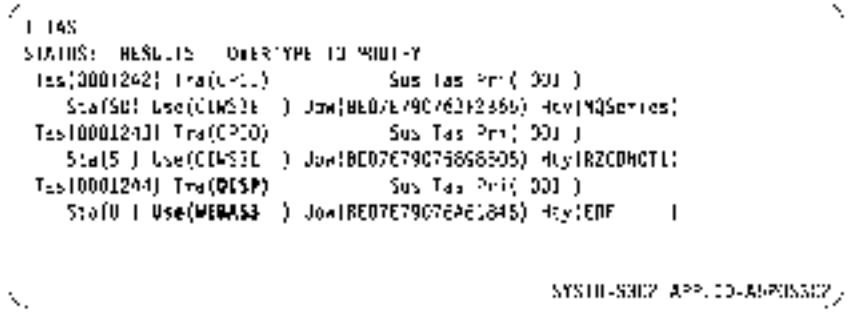


Figure 8-24 DISP transaction running in CICSS3C2

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000322

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000323

9

Security scenarios using CICS WS-Security support

In this chapter we outline several security scenarios that demonstrate how you can secure CICS Web services using the CICS-supplied message handler, DFHWSSE1.

We provide step-by-step security configuration for a number of scenarios, including:

- Basic authentication
- Signing a SOAP message
- Encrypting a SOAP message

Note: The security scenarios documented in this chapter use the CICS-supplied WS-Security support. See Chapter 8, "Security scenarios" on page 247 for WS-Security scenarios using custom message handlers.

We show how we configured both CICS and WebSphere Application Server for these security scenarios.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000324

9.1 Preparation

Security techniques for securing Web services were discussed in Chapter 7, "Securing Web services" on page 213. In this chapter, we describe the following security scenarios:

- Basic authentication

  In this scenario we show how the CICS-supplied message handler DFHWSSE1 can extract a WS-Security UsernameToken from a SOAP header, validate the username and password, and set the user ID of the CICS task to the username passed in the header. This scenario is described in "Basic authentication" on page 302.

- Signing a SOAP message

  In this scenario we show how CICS and WebSphere Application Server can be configured to exchange signed SOAP messages. The SOAP message sent by WebSphere Application Server contains an X.509 certificate that allows the CICS transaction to run under a user ID associated with the X.509 certificate. See "Signing a SOAP message" on page 330 for details of this scenario.

- Encrypting a SOAP message

  In this scenario we show how CICS and WebSphere Application Server can be configured to exchange encrypted SOAP messages. The SOAP message contains an encrypted body so that the content of the message cannot be understood by an unauthorized party. This scenario is described in 9.6, "Encrypting a SOAP message" on page 357.

9.1.1 Software checklist

The software we used is listed in Table 9-1.

Table 9-1 Software used in the WS-Security security scenarios

Windows	z/OS
Internet Explorer V6.0	z/OS V1.7
Windows XP V5.1 SP2	CICS Transaction Server V3.1 (with PTFs UK15271 and UK15261 installed)
IBM WebSphere Application Server - ND V6.1.0.0	RACF V1.7

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000325

Windows	z/OS
Our J2EE applications ► CatalogSec2.ear Catalog manager service requester application with no security enabled ► CatalogSec2_WS-Security_BasicAuth.ear Catalog manager service requester application with WS-security basic authentication enabled ► CatalogSec2_WS-Security_Signature.ear Catalog manager service requester application with WS-security signature enabled ► CatalogSec2_WS-Security_Encryption.ear Catalog manager service requester application with WS-security encryption enabled	Our user-written CICS programs ► CIWSMSGH (message handler program) This program changes the transaction ID for the Web service requests received in the pipeline. ► SNIFFER (message handler program) This program browses through the containers available in the pipeline

For this chapter we use a slightly different version of the Catalog application than the version used in Chapter 8. In the wsdl created by CICS the names of the Web service binding ports are the same for each of the three Web services. We noticed that this caused some problems in AST when editing the WebSphere WS-Security configuration files. We changed the name of the binding ports in the corresponding wsdl files from the default DFH0XCMNPort to DFH0XCMNPortPO for the placeOrder service, DFH0XCMNPortIS for inquireSingle and DFH0XCMNPortIC for inquireCatalog. We then recreated the Web services using our development tool (Rational Application Developer).

9.1.2 Definition checklist

The definitions we used are listed in Table 9-2

Table 9-2 Settings used in the WS-Security security scenarios

Value	CICS TS	WebSphere Application Server
IP name	wtsc.tss.ibm.com	mikee31
IP address	9.12.4.75	9.12.4.217
TCP/IP port	14301	9081
Jobname	CIWSS3C1	
APPLID	MQPOS3C1	
TCPIPSERVICE	S3C1	

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000326

Value	CICS TS	WebSphere Application Server
Provider PIPELINEs	PIPE1 used for inquireSingle and inquireCatalog services PIPEWSSE used for placeOrder service	
Configuration files	ITSO_7206_basicsoap12provider.xml ITSO_7206_wssec_basicauth_provider.xml ITSO_7206_wssec_signature_provider.xml ITSO_7206_wssec_encryption_provider.xml	
URIMAP	SECPORDR SECICATA SECISING	

The user IDs we used in our configuration are listed in Table 9-3.

Table 9-3 User IDs

Value	CICS TS
CICS region user ID	CIWS3D
CICS default user ID	CICSUSER
User IDs for which we wish to permit access	CIWSNW WEBAS1

Because the Catalog application inquireSingle and inquireCatalog services are read-only services, whereas the placeOrder service updates the database, we show here how to secure the placeOrder service. We run the inquireSingle and inquireCatalog services under a pre-defined user D (CIWSNW). The placeOrder service is run under the caller's user ID (WEBAS1).

Tip: For the examples described in this book, we permit access for single user IDs. In a production environment, you will probably create a group of users requiring common access. Once a group is built, you can permit access for the group. This allows users' access to be controlled by the group to which they belong, rather than by individual permissions. This simplifies the security definitions required.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000327

9.2 Basic security configuration

First we discuss our basic security configuration, taking a CICS region with no security and configuring it to enable transaction security (we do not implement other types of CICS security such as resource security and command security).

Note: The security scenarios in this chapter are based on a different MVS system that those documented in Chapter 8, "Security scenarios" on page 247. We therefore repeat some of the security setup steps for CICS region CIWSS3C1 here.

In this section, we document the following steps:

- Creating a RACF key ring
- System initialization table parameters necessary to set up basic CICS security
- Testing the basic security configuration
- Configuring the pipeline
- Setting a user ID on a URIMAP definition

9.2.1 Creating a RACF key ring

In order to use the CICS WS-Security support for signing and encryption, it is necessary to create a RACF key ring, public-private key pairs, and X.509 certificates. In 9.4, "Certificate and key pair generation" on page 319 we show how to create certificates and key pairs for the signing and encryption scenarios. In the basic CICS security setup, however, we created a key ring for use by CICS region CIWSS3C1.

Note: In our tests we found that the CICS region must be connected to a RACF key ring even if the only CICS WS-Security support being used is basic authentication.

The sample clist DFH$RING is provided by CICS TS to help you build a RACF key ring and sample X.509 certificates for SSL usage. We used the DFH$RING clist to create the key ring `Ciws.Ciwss3cl`. See 8.5.1, "Creating a key ring and certificates on z/OS for CICS" on page 274 for information on using the DFH$RING sample clist.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000328

9.2.2 Specifying the security SIT parameters

We specified the following security SIT parameters for our CICS region:

- KEYRING=CICS.CICSSDC

  This specifies the name of a key ring we created in the RACF database that contains keys and certificates used by this CICS region.

- SEC=YES

  SEC=YES was specified to indicate that we wanted RACF services to control access to CICS resources.

- SECPRFX=YES

  We used security prefixing (SECPRFX=YES) in our CICS region, which prevents our RACF security profiles from affecting other CICS regions. This is useful in a production environment since it means all security profiles are unique to an individual region; however, it can mean more work for the security administrator because more profiles must be defined.

- XTRAN=YES

  XTRAN=YES was specified so CICS would control who could start transactions.

- XUSER=YES

  XUSER=YES specifies that CICS is to perform surrogate user checking.

For detailed information on CICS security, see *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454.

9.2.3 Testing the basic security configuration

In our testing, we followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec2.ear.

When attempting to place an order with the catalog application, the Web browser received an error message and the cause of the error was reported by CICS as a security violation (Example 9-1).

Example 9-1 CICSUSER security violation messages during CPIH

```
DFHXS1111 08/09/2006 18:05:38 46P053CI CPIH Security violation by user CICSUSER for resource
          CICS3D.CPIH in class TCICSTRN. SAF codes are (X'0CCCCCC8',X'00000008'), ESM codes
          are (X'CCCCCCC8',X'00000000').
DFHWB0561 08/09/2006 18:05:38 46P053CI An attempt to attach a CICS Web alias transaction for
          userid CICSUSER has failed because the user is not authorized to execute transaction
          CPIH. Host IP address: 9.12.4.75. Client IP address: 9.12.4.201.
          TCPIPSERVICE: SSCI.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000329

```
DFHXS2003 09/09/2006 18:05:38 A6P053CL Security violation has been detected term id = ????,
          trans id = CPIH, userid = CICSUSER
```

The CICS default user ID does not have access to the CPIH transaction. This shows that the CICS default user ID is not authorized to access the placeOrder service. In the following scenarios, we enable placeOrder service access to a set of authorized user IDs.

9.2.4 Configuring the pipeline

Example 9-2 shows the inital pipeline configuration file used for our CICS WS-Security test setup

Example 9-2 Pipeline configuration file ITSO_7206_basicsoap12provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/hts/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/hts/cics/pipeline
  provider.xsd ">
    <transport>
      <default_transport_handler_list>
          <handler>
             <program>CIWSMSGH</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
    </transport>
    <service>
      <service_handler_list>
        <handler>
          <program>SHOFFER</program>
          <handler_parameter_list/>
        </handler>
      </service_handler_list>
      <terminal_handler>
        <cics_soap_1.2_handler/>
      </terminal_handler>
    </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

- The message handler program CIWSMSGH is used to change the transaction ID to INQS, INQC, or ORDR based on the service request. See "Customizing the pipeline configuration file" on page 80 for more information on the CIWSMSGH program.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000330

- The message handler program SNIFFER is a simple program that browses through the containers available in the pipeline. See "Using SNIFFER" on page 106 for more information on the SNIFFER program.

9.2.5 Setting a user ID on a URIMAP definition

We initially want to run the CPIH, INQS, INQC, and ORDR transactions under a pre-defined user ID, other than the default CICS user ID for the security scenarios that follow, we then show how to run ORDR transactions for specific placeOrder service requests under user IDs which flow in the SOAP messages

To set the pre-defined user ID for the CPIH and ORDR transactions, we created a URIMAP definition for the placeOrder service such that the transactions run with the user ID CICSNW. See Section 8.3, "Setting the user ID on a URIMAP definition" on page 252 for more information on defining URIMAPs. We then permitted RACF access to the CPIH and ORDR transactions for the user ID CICSNW.

Figure 9-1 shows the CPIH and ORDR transactions running under user ID CICSNW

```
INQUIRE TASK
STATUS:  RESULTS - OVERTYPE TO MODIFY
  Tas(0000082) Tra(CPIH)           Sta Tas Pri( 001 )
     Sta(U ) Use(CICSNW  ) Dsn(B=3BFA3A452A25D3C) Hty(DFHPO**)
  Tas(0000083) Tra(ORDR)           Sta Tas Pri( 001 )
     Sta(U ) Use(CICSNW  ) Dsn(B=3BFA3A474A2D9C9) Hty(EC*     )
  Tas(0000084) Tra(CEUT) Fac(EC22) Sta Ter Pri( 001 )
     Sta(SD) Use(WISCLM  ) Dsn(B=3BFA3A47A210343) Hty(ZCICWA**)
  Tas(0000085) Tra(CONT) Fac(CC21) Sta Ter Pri( 255 )
     Sta(TO) Use(WISCLM  ) Dsn(B=3BFA3A5A97D2631)
```

Figure 9-1 ORDR executing with preset user ID

We also created URIMAP resources for the inquireSingle and inquireCatalog services such that the INQS and INQC transactions also run with user ID CICSNW, and we permitted RACF access to the INQS and INQC transactions for the user ID CICSNW.

9.3 Basic authentication

WebSphere Application Server supports the generation of security tokens that are passed in the WS-Security SOAP header. A UsernameToken is a token for

Document ID: [illegible] Page 00450

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000331

basic authentication; it has a user name and password. In 8.4, "Enabling SOAP message security with HTTP" on page 257 we show how we wrote the message handler program CIWSSECS, which we then used to process a UsernameToken passed in a Web service call from WebSphere Application Server.

In this section we show how a UsernameToken can also be processed by the CICS-supplied message handler, DFHWSSE1. DFHWSSE1 extracts the UsernameToken from the SOAP header, validates the username and password, and sets the user ID of the CICS task to the username passed in the header.

Note: CICS supports UserNameToken in service provider mode only.

This scenario is shown in Figure 9-2.

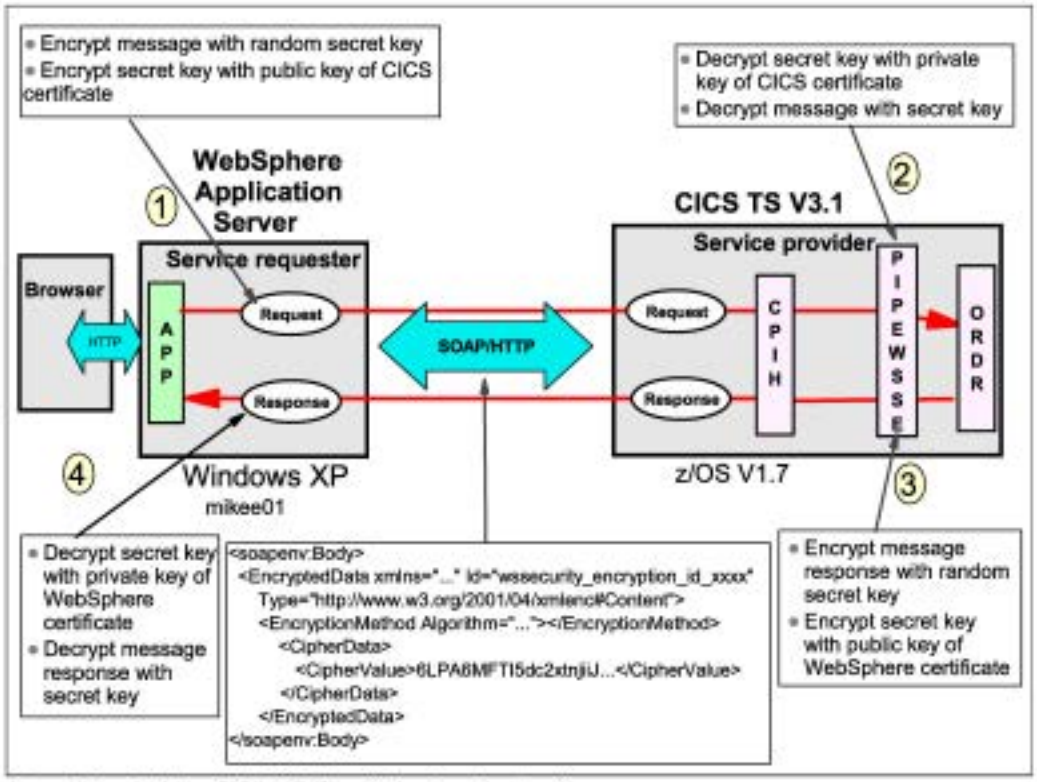


Figure 9-2 Enabling CICS WS-Security basic authentication

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 332/f9

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000332

Recommendation: Internal tests have shown that the performance of a user-written message handler program such as CIWSSECS for processing a UsernameToken is significantly better than DFHWSSE1. If throughput is expected to be high and performance is a priority, we recommend that you use a user-written message handler program for basic authentication. If, however, you do not want to write custom handlers, or if performance is not a priority, then enabling basic authentication using the CICS-supplied DFHWSSE1 is a simpler solution.

We document the following steps that we performed to enable support for basic authentication:

- Configuring the service requester for basic authentication
- Configuring CICS for basic authentication
- Testing the basic authentication scenario

9.3.1 Configuring the service requester for basic authentication

The service requester setup tasks that we cover here are as follows:

- Defining the WebSphere WS-Security constraint for the Web service client
- Re-deploying the Web service client application

WebSphere WS-Security configuration files

The WebSphere WS-Security constraints are defined in the IBM extension of the J2EE Web services deployment descriptor. There are four configuration files:

- Client deployment descriptor extension file - includes request generator and response consumer constraints:

  ibm-webservicesclient-ext.xmi

- Client binding configuration file - includes how to apply request generator and response consumer constraints:

  ibm-webservicesclient-bnd.xmi

- Server deployment descriptor extension file - includes request consumer and response generator constraints:

  ibm-webservices-ext.xmi

- Server binding configuration file - includes how to apply request consumer and response generator constraints:

  ibm-webservices-bnd.xmi

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000333

The deployment descriptor extension files specify *what* security constraints are required, for example, what type of security token and whether to sign the message. The binding files specify *how* to apply the required security constraints defined in the deployment descriptor extension, for example, which security token is inserted and which keys are used for signing.

These deployment descriptor extension and binding files define the application-level security constraints and they belong to each application. See "WebSphere and SOAP message security" on page 239 for more information on configuring WebSphere WS-Security.

For our scenario, we needed to configure the client deployment descriptor and the client binding only, since our service provider is running in CICS and not in WebSphere Application Server.

Importing the base application

To configure our Web Service client to use WS-Security, we used the IBM WebSphere Application Server Toolkit V6.1 (AST).

Note: You can also use Rational Application Developer V6.0 (RAD) to configure WS-Security for WebSphere applications.

We performed the following steps:

1. We opened the AST in a new workspace. In Windows we clicked:

   Start → Programs → IBM WebSphere → Application Server Toolkit V6.1 → Application Server Toolkit

   We specified a new workspace for this scenario called

   `C:\AST\CatalogSec2\CatalogSec2BasicAuth`

2. To import the CatalogSec2 base application we clicked the **File** menu and then **Import**.
3. We selected **EAR file** in the list of options and clicked **Next**.
4. We clicked **Browse** to select the CatalogSec2.ear file. This file contains the Catalog base application.
5. We also changed the EAR project name to CatalogSec2BasicAuth as shown in Figure 9-3.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000334



Figure 9-3 Importing the CatalogSec2 ear file

6. Finally, we clicked **Next** → **Next** → **Finish**.

Editing the client configuration

We performed the following steps to edit the client configuration:

1. We expanded the Dynamic Web Project (CatalogSec2Web) in the Project Explorer and opened (double-clicked) the deployment descriptor as shown in Figure 9-4.

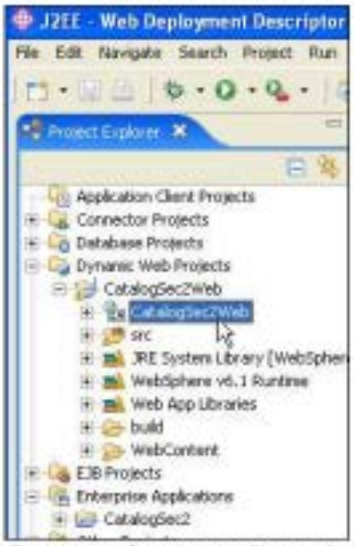


Figure 9-4 Opening the CatalogSec2Web Deployment Descriptor.

2. After the Deployment Descriptor Editor opened, we selected the **WS Extension** page. The WS Extension page is used for editing the client's

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 335/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000335

deployment descriptor extension file (ibm-webservicesclient-ext.xmi), so you can specify what security is required. Figure 9-5 shows the WS Extension page in the Deployment Descriptor Editor.

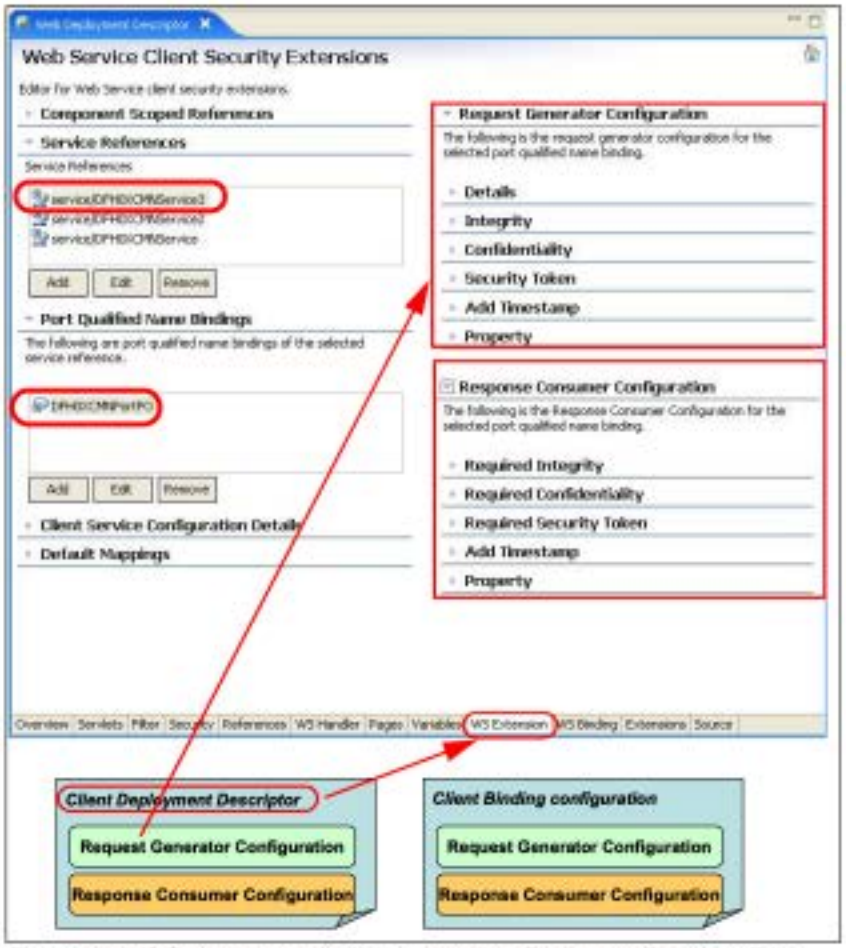


Figure 9-5 WS Extension page in Web service client Deployment Descriptor Editor

3. We clicked the **service/DFH0XCMNService3** reference (for the placeOrder service) and **DFH0XCMNPortPO** (the port binding for the service reference).
4. In the Request Generator Configuration, we expanded **Security Token** and clicked **Add**. We entered the name `basicauth`.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 336/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000336

5. We selected the **Username Token** token type from the drop-down list. When you select a Token type, the Local name is filled in automatically (Figure 9-6). We left the URI empty.



Figure 9-6 Security Token Dialog for specifying basic authentication

6. We clicked **OK** and a security token was created. We then saved the configuration by pressing Ctrl+s.
7. After specifying the security token, a corresponding token generator must be specified in the binding configuration. The WS Binding page is for editing the client's binding file, so you can specify how to apply the required security. We clicked the **WS Binding** tab. Figure 9-7 shows the WS Binding page in the Deployment Descriptor Editor.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 337/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000337

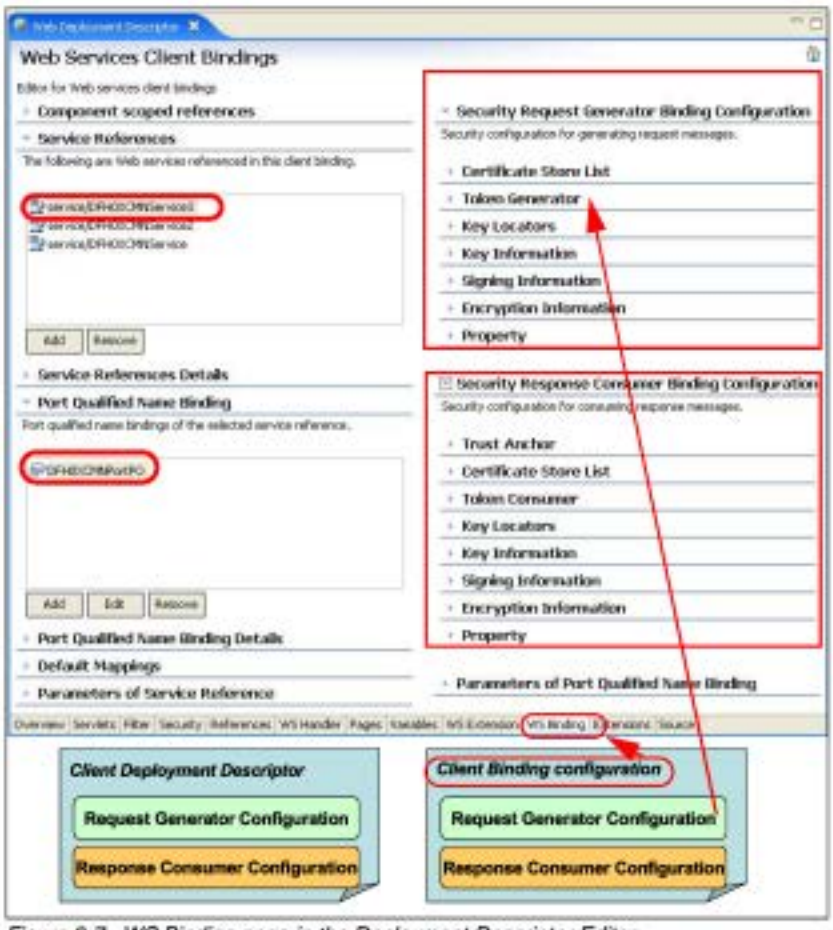


Figure 9-7 WS Binding page in the Deployment Descriptor Editor

8. We selected **Token Generator** and clicked **Add**. We entered a Token generator name `basicauthToken` and allowed the Token generator class to default as `com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator`
9. To select the security token, we expanded the drop-down list and selected the **basicauth** security token that we defined previously on the WS Extension page.
10. We checked **Use value type** and selected the **Username Token** value type from the drop-down list. This selection filled the local name and callback handler.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 338/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000338

11. We entered the user ID WEBAS1 and password REDBOOKS that we wanted to use to identify the Web service client (Figure 9-8).

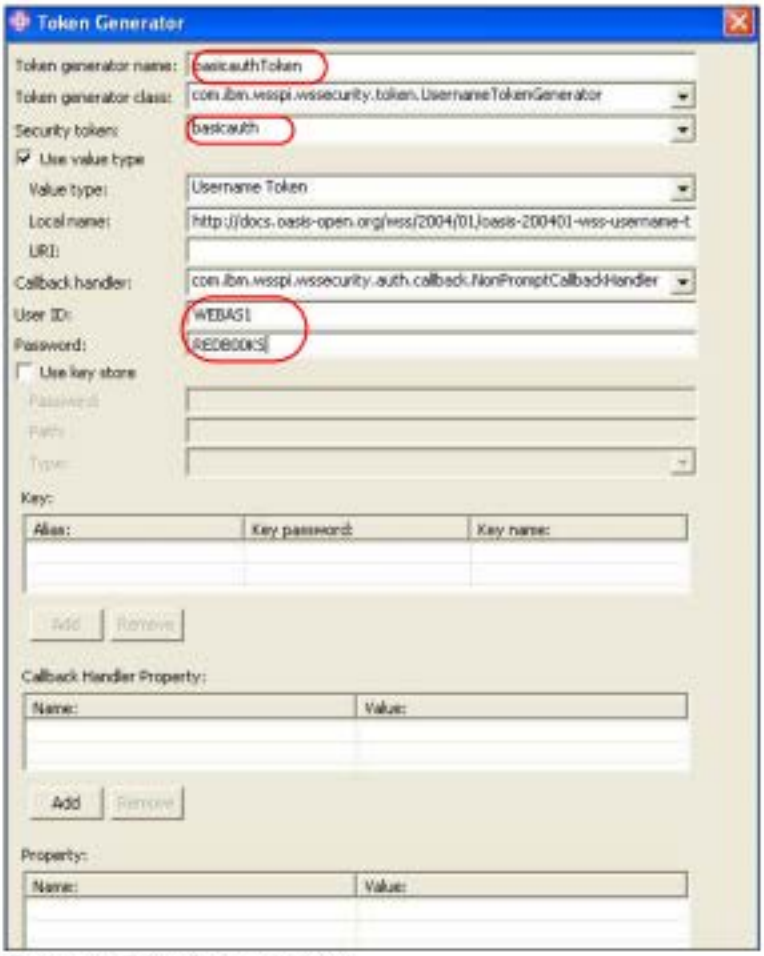


Figure 9-8 Token Generator dialog

12. We clicked **OK**, and the token generator was created. We then saved the configuration.

Note: The User ID and Password in Figure 9-8 represent the credentials of the application server, *not* the browser user. The same credentials are passed to CICS in the SOAP message irrespective of the identity of the end user.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 339/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000339

Redeploying the Web service application

After configuring a basic authentication security constraint for the service requester application, we exported a new EAR file called CatalogSec2_WS-Security_BasicAuth.ear. We followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec2_WS-Security_BasicAuth.ear.

For further information on configuring WS-Security in WebSphere, refer to *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257.

9.3.2 Configuring CICS

The credentials configured using the Application Server Toolkit are used to form the WS-Security header that is contained in the SOAP message. An example of this message is shown in Example 9-3.

Example 9-3 SOAP security header extracted from a SOAP message

```
<wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">
    <wsse:UsernameToken>
      <wsse:Username>WEBAS1</wsse:Username>
      <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401
       -wss-username-token-profile-1.0#PasswordText">REDBOOKS</wsse:Password>
    </wsse:UsernameToken>
</wsse:Security>
```

This header includes the mustUnderstand="1" attribute, which indicates that either this header must be processed or a SOAP fault thrown.

In order for CICS to process the security token in the SOAP header of the placeOrder service request, we made the following changes to the CICS configuration:

- Created a pipeline configuration file for basic authentication
- Created a new pipeline to process placeOrder requests and altered the URIMAP for the placeOrder service to use the new pipeline
- Permitted access to the ORDR transaction for user ID WEBAS1
- Permitted surrogate access for the user ID WEBAS1

The details for those procedures are presented in the following sections

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000340

Creating a pipeline configuration file for basic authentication

Example 9-4 shows the basic authentication pipeline configuration file that includes the message handler DFHWSSE1, and the configuration information for the handler.

Example 9-4 Pipeline config file, ITSO_7206_wsse_basicauth_provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/txseries/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/txseries/cics/pipeline
  provider.xsd ">
    <transport>
      <default_transport_handler_list>
          <handler>
             <program>CWSMSG</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
    </transport>
    <service>
      <service_handler_list>
        <handler>
          <program>SMI0000</program>
          <handler_parameter_list/>
        </handler>
        <wsse_handler>
          <dfhwsse_configuration version="1">
            <authentication mode="basic">
            </authentication>
          </dfhwsse_configuration>
        </wsse_handler>
      </service_handler_list>
      <terminal_handler>
        <cics_soap_1.2_handler/>
      </terminal_handler>
    </service>
  <appHandler>DFHPITP</appHandler>
</provider_pipeline>
```

The <wsse_handler> element shown in Example 9-4 contains a <dfhwsse_configuration> element that specifies configuration information for DFHWSSE1.

The <authentication mode="basic"/> specifies that inbound messages must contain a UserNameToken.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000341

Creating a new pipeline to process placeOrder requests

Since the inquireCatalog and inquireSingle service requests will not be subject to basic authentication, we need to create a new pipeline specifically for the placeOrder requests. We also need to change the SECPORDR URIMAP definition to link the placeOrder service to the new pipeline. We do this using the RDO commands in Example 9-5 and then re-installing the PIPELINE and URIMAP resource definitions.

Example 9-5 Create pipeline for WS-Security basic authentication

```
CEDA COPY PIPELINE(PIPE1) GROUP(SSCTEXMS) AS(PIPEWSSE)
CEDA ALTER PIPELINE(PIPEWSSE) GROUP(SSCTEXMS)
configfile(/CICS/SSCT/configs/CICS_FZIB_wssec_basicauth_provider.xml)
CEDA ALTER URIMAP(SECPORDR) GROUP(SSCTEXMS) PIPELINE(PIPEWSSE)
```

Permitting access to the ORDR transaction

We permitted access to the ORDR transaction for the user ID WEBAS1 with the following:

```
PERMIT CICS3D.ORDR CLASS(TCICSTRN) ID(WEBAS1) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Permitting surrogate access

Example 9-6 shows the RACF commands that we used to permit the pre-defined user ID (CICSNW) to start transactions (such as ORDR) with the user ID WEBAS1.

Example 9-6 RACF commands to allow CICSNW to act as surrogate for WEBAS1

```
RDEFINE SURROGAT WEBAS1.DFHSTART UACC(NONE) OWNER(MAINT)
PERMIT WEBAS1.DFHSTART CLASS(SURROGAT) ID(CICSNW) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

For more information regarding surrogate user security, see *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454.

9.3.3 Testing basic authentication

Now when we submit an order request on the Web browser, the credentials are added to the SOAP Security header and are extracted by the DFHWSSE1 program.

Figure 9-9 shows the ORDR transaction running with the user ID WEBAS1, while the CPIH transaction continues to execute with the CICSNW user ID.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000342

```
INQUIRE TASK
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tas(0000055) Tra(CPIH)           Sus Tas Pri( 001 )
    Sta(U ) Use(CIWSMW  ) Uow(BF3C2E9E2114284E) Hty(RZCBNOTI)
 Tas(0000056) Tra(ORDR)           Sus Tas Pri( 001 )
    Sta(U ) Use(WEBAS1  ) Uow(BF3C2E9E24801B8B) Hty(EDF     )
 Tas(0000058) Tra(CEDF) Fac(E024) Sus Ter Pri( 001 )
    Sta(SD) Use(NIGELW  ) Uow(BF3C2E9E28D4A149) Hty(ZCIOWAIT)
 Tas(0000059) Tra(CEMT) Fac(E025) Run Ter Pri( 255 )
    Sta(TO) Use(NIGELW  ) Uow(BF3C2EA64198CCCE)

                                                SYSID=S3C1 APPLID=A6POS3C1
```

Figure 9-9 ORDR executing with basic authentication user ID

SOAP fault messages

In this section we show some of the SOAP faults that you may see after enabling CICS WS-Security basic authentication.

- Service requester does not send UserNameToken

  If the service requester does not send a UsernameToken, DFHWSSE1 returns a SOAP fault as shown in Figure 9-10.

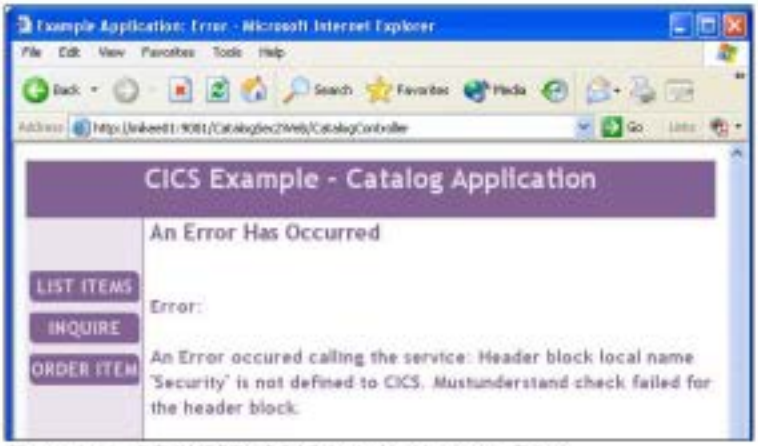


Figure 9-10 DFHWSSE1 SOAP fault: No UsernameToken

- UserNameToken credentials are invalid

  If the credentials extracted do not validate, then DFHWSSE1 creates a SOAP fault message as shown in Figure 9-11.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 343/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000343

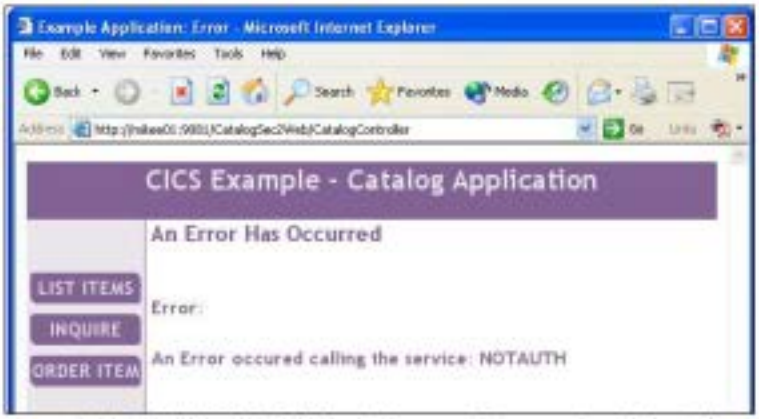


Figure 9-11 DFHWSSE1 SOAP fault: UsernameToken credentials invalid

The SOAP fault for this message, shown in Example 9-7, highlights that the user credentials are invalid and that the RACF revoke count is incremented for this user ID.

Example 9-7 SOAP fault for invalid basic authentication credentials

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>wsse:FailedAuthentication
      </faultcode>
      <faultstring>NOTAUTH
      </faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
          <message>The supplied password is wrong. If the external security
          manager is RACF, the revoke count maintained by RACF is incremented
          </message>
          <errorcode>3
          </errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 344/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000344

- WebSphere rejects SOAP response sent by CICS due to empty security header.

  During our testing we experienced the problem shown in Figure 9-12.

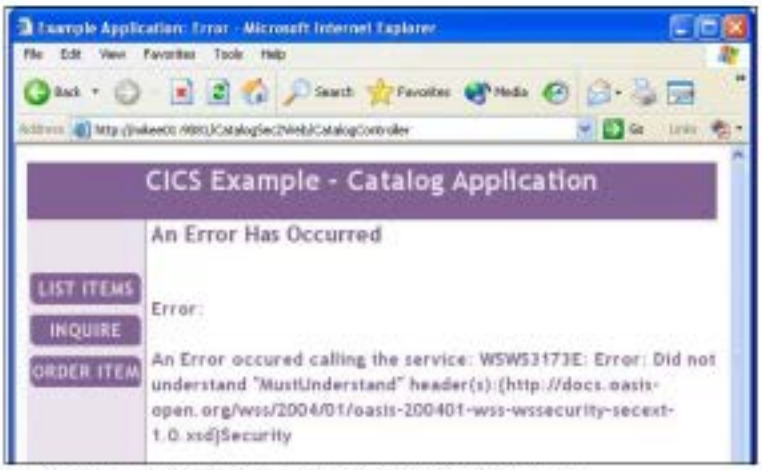


Figure 9-12 DFHWSSE1 SOAP fault: Empty security token

In this case CICS has returned the empty security header shown in Example 9-8.

Example 9-8 Empty security header returned from CICS

```
<wsse:Security
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" SOAP-ENV:mustUnderstand="1"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
tility-1.0.xsd" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
```

We found that, depending on the version of WebSphere Application Server and the maintenance level of CICS TS V3.1, it is possible that WebSphere will reject this SOAP response from CICS due to it containing an empty security header.

The WebSphere exception for this error, shown in Example 9-9, highlights that WebSphere is not expecting a security header and is rejecting the empty one sent by CICS because it contains the `mustUnderstand="1"` attribute.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 345/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000345

Example 9-9 WebSphere exception for empty security header

```
Exception: WSWS3173E: Error: Did not understand "MustUnderstand"
header(s):{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-s
ecext-1.0.xsd}Security:
```

We were able to bypass this problem by configuring WebSphere to optionally receive a security token sent in the CICS response.

Perform the following steps to configure WebSphere to tolerate an empty security header sent in the CICS response:

1. Open the CatalogSec2Web Deployment Descriptor.
2. Select the WS Extension page.
3. In Service References select the **service/DFH0XCMNService3** reference. then click **DFH0XCMNPortPO** in Port Qualified Name Bindings.
4. Expand **Response Consumer Configuration**.
5. Expand **Required Security Token** and click **Add**.
6. In the Required Security Token window returned (Figure 9-13), make the following entries and selections:
   - Enter a name for this required security token, for example, `requntoken`.
   - Select **Username Token** from the Token type drop-down list.
   - Select **Optional** for the Usage type from the drop-down list.
     The available choices are Required or Optional. If you select Required, a SOAP fault is thrown if a required security token is not included in a client's request message. If you select Optional, the process of consuming a request message continues without throwing a SOAP fault.
7. Click **OK**.



Figure 9-13 WS Extension page Required Security Token dialog box

8. Save the configuration by pressing Ctrl+s.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 346/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000346

9. Open the WS Binding page.
10. In Service References select the **service/DFH0XCMNService3** reference, then click **DFH0XCMNPortPO** in Port Qualified Name Bindings.
11. Expand **Security Response Consumer Binding Configuration.**
12. Expand **Token Consumer** and click **Add.**
13. In the Token Consumer dialog box returned (Figure 9-14) make the following entries and selections:
    - Enter a Token consumer name, for example `con_untcon`.
    - Select **com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer** as the Token consumer class.
    - From the Security token drop-down list select the security token specified in the WS Extension page. In our case this was **requntoken**.
    - Select the **Use jaas.config** option and enter `system.wssecurity.UsernameToken` as the jaas.config name.
14. Click **OK**.

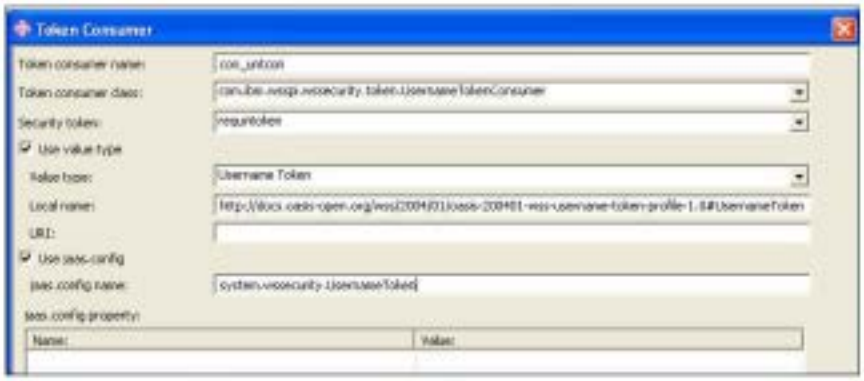


Figure 9-14 Token Consumer dialog box in WS Binding page

15. Save the Deployment Descriptor.
16. Export the application as an ear file and install it in the WebSphere Application Server.

Important: APAR PK31924 has been raised for this problem. If you experience this problem and no fix is available for PK31924 at the time, you will need to configure WebSphere to tolerate an empty security header sent by CICS. This problem can also occur for digital signature processing.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 347/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000347

9.4 Certificate and key pair generation

In this section we show how we created the certificates and key pairs that are used in the WS-Security signature and encryption test scenarios that follow.

When setting up a WS-Security environment you have a choice between self-signed certificates and CA-signed certificates:

- Self-signed certificates

  A self-signed certificate is an identity certificate that is signed by its creator, so the creator is verifying that the certificate is valid.

- Certificating authority (CA) signed certificates

  CA-signed certificates are created by a user organization and sent to a CA to be signed. Before signing a certificate the CA verifies that the organization requesting the certificate is actually who they claim to be, so the CA is verifying that the certificate is valid.

In the examples in this chapter, we use self-signed certificates because they are more easily and quickly generated. In a production environment, it is expected that CA-signed certificates will be used because they provide a more secure solution.

Figure 9-15 shows the certificates and key pairs used in the signature and encryption test scenarios.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000348

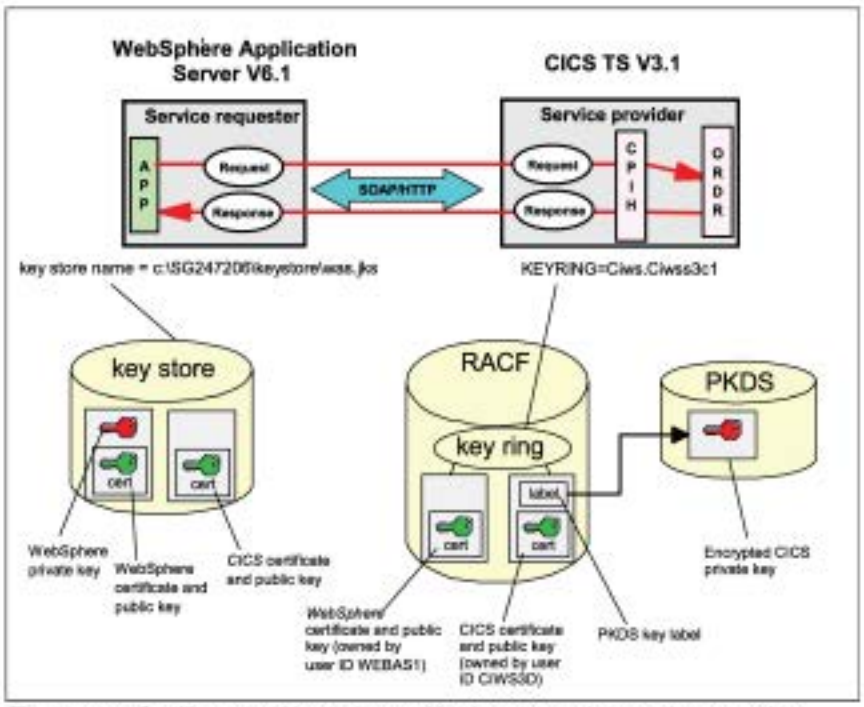


Figure 9-15 Certificates and key pairs used in signature and encryption scenarios

Figure 9-15 shows two stores for our certificates and key pairs:

- The WebSphere key store
  - The WebSphere key store contains the WebSphere certificate and holds the WebSphere public and private key pair.
  - The key store also contains the CICS certificate with the public key only.
- RACF
  - RACF contains the CICS certificate and holds the CICS public key. The associated private key is stored in a Private Key Data Set (PKDS).

    In a public key cryptographic system, it is a priority to maintain the security of the private key. It is vital that only the intended user or application have access to the private key.

    ICSF and the cryptographic hardware features ensure this by enciphering PKA private keys under a unique PKA object protection key. The PKA object protection key has itself been enciphered under a PKA *master key*. Each PKA private key also has a name that is cryptographically bound to

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 349/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000349

the private key and cannot be altered. ICSF uses the private key name or the *PKDS key label* to control access to the private key. This combination of hardware-enforced coupling of cryptographic protection and access control, through the use of RACF, is unique to ICSF. It provides a significant level of security and integrity for PKA applications.

– RACF also contains the WebSphere certificate with the public key only.

Note: If you use CA-signed certificates, you also need to add the CA certificates to the WebSphere key store and RACF key ring.

In this section, we document the following steps:

- Validating the cryptography hardware environment
- Creating the CICS certificate and key pair
- Creating the key store, WebSphere certificate, and key pair
- Adding the CICS certificate to the key store
- Adding the WebSphere certificate to RACF

Validating the cryptography hardware environment

CICS support for WS-Security signature and encryption processing benefits from the hardware acceleration capabilities of System z. It is a requirement, therefore, to run CICS on a system that has the appropriate cryptographic hardware configured. The specific cryptographic hardware requirement is dependent on the server type and z/OS level. See 6.12.1, "Cryptographic hardware requirements for CICS WS-Security" on page 211 for more information on cryptographic hardware requirements.

The SC66 host system used for our signature and encryption test scenarios was a z9 EC (Enterprise Class) with 18 CPs. The SC66 had the following cryptographic hardware functions enabled:

- CP Assist for Cryptographic Function (CPACF)
- Crypto Express2 (CEX2) feature

Three CEX2C coprocessors and one CEX2A accelerator were configured.

Important: The CEX2 feature requires ICSF to be active.

Integrated Cryptographic Service facility

The Integrated Cryptographic Service Facility (ICSF) is a software element of z/OS that works with hardware cryptographic features and RACF to provide

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 350/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000350

secure, high-speed cryptographic services in the z/OS environment. See 6.12, "Integrated Cryptographic Service Facility" on page 210 for more information.

Example 9-10 shows the startup messages of ICSF on the SC66 system. It shows that 3 CEX2C coprocessors and 1 CEX2A coprocessor are configured.

Example 9-10 ICSF startup messages

```
S CSF
IEF695I START CSF WITH JOBNAME CSF IS ASSIGNED TO USER STC, GROUP SYST
IEE402I CSF    STARTED   ASID=0096   SC66
CSFM441I CRYPTO EXPRESS2 COPROCESSOR E01, SERIAL NUMBER 94000264, ACTIVE
CSFM441I CRYPTO  EXPRESS2 COPROCESSOR E02, SERIAL NUMBER 95001434, ACTIVE
CSFM441I CRYPTO EXPRESS2 COPROCESSOR E03, SERIAL NUMBER 95001437, ACTIVE
CSFM135I CRYPTO EXPRESS2 ACCELERATOR F00 IS ACTIVE
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
```

ICSF must be configured and started in order to create public/private key pairs to be used by CICS for XML signature or encryption processing.

Creating the CICS certificate and key pair

In order to enable signature and encryption processing, we need to create a certificate and key pair to be used by CICS, and then connect the certificate to the CICS key ring.

Example 9-11 shows the RACF command that we used to create the CICS X.509 certificate.

Example 9-11 Create the CICS X.509 certificate (CICSCERT)

```
RACDCERT ID(CICSSOJ) GENCERT
 SUBJECTSDN(CN('CICSCERT')
            T('CICSSOJ-cert')
            OU('PSSC')
            O('ITSO')
            L('Endicott')
            SP('New York')
            C('US'))
 NOTBEFORE(DATE(2005-01-01) TIME(00:00:00))
 NOTAFTER(DATE(2010-12-31) TIME(23:59:59))
 KEYUSAGE(DOCSIGN DATAENCRYPT)
 WITHLABEL('CICSCERT')
 SIZE(1024)
 TCSF
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000351

- The RACDCERT GENCERT command creates the certificate and a public/private key pair.
- ID specifies that the CICS region user ID CIWS3D is to be associated with the certificate.
- SUBJECTDSN specifies the subject's X.509 distinguished name.
- NOTBEFORE and NOTAFTER specify the certificate validity range.
- KEYUSAGE specifies how the keys associated with the certificate are to be used. We specify:
  - DOCSIGN specifies that the certificate will be used for signing.
  - DATAENCRYPT specifies that the certificate will be used for encryption.
- WITHLABEL specifies the label name to be associated with the certificate.
- SIZE specifies the size of the private key expressed in decimal bits. We specified a high strength key length of 1024.
- ICSF specifies that the private key to be generated is an RSA key to be stored in the ICSF PKDS (public key data set).

  A certificate that CICS uses to sign SOAP messages (or decrypt encrypted SOAP messages) must be created with the ICSF option and ICSF must be configured and started in order for the certificate to be created or used by CICS. If ICSF is not operational, or does not have access to any cryptographic coprocessors or accelerators, you will receive the message `ICSF is not operational` when attempting to create the X.509 certificate.

Important: A certificate that CICS uses to sign SOAP messages, or to decrypt encrypted SOAP messages, must be created with the ICSF option.

Example 9-12 shows the RACF commands that we used to connect the CICS X.509 certificate to the key ring and to list the certificate.

Example 9-12 RACF command to connect CICS certificate to RACF key ring

```
RACDCERT ID(CIWS3D) CONNECT(ID(CIWS3D) LABEL('CICSCERT') RING(Ciws.Ciwss3cl))
RACDCERT ID(CIWS3D) LIST
```

Example 9-13 shows the output of the RACDCERT LIST command.

Example 9-13 Listing of CICS X.509 certificate (CICSCERT)

```
Digital certificate information for user CIWS3D:

  Label: CICSCERT
  Certificate ID: 2QbDyebi88TDycP1w8XZ40BA
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Page 352/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000352

```
Status: TRUST
Start Date: 2005/01/01 00:00:00
End Date:   2014/12/31 23:59:59
Serial Number:
     >00<
Issuer's Name:
     >CN=CICSCERT.1 Created cert.OU=PSSC.O=IISO... ENDCERT.SP=NEW YORK,C=US
     <
Subject's Name:
     >CN=CICSCERT.1 Created cert.OU=PSSC.O=IISO...ENDCERT.SP=NEW YORK,C=US
     <
Key Usage: DATAENCRYPT, DOCSIGN
Private Key Type: ICSF
Private Key Size: 1024
PKDS Label: IRR.DIGTCERT.CCU530.SC00.BF3B6F434ED31FC9
Ring Associations:
  Ring Owner: CICSSC
  Ring:
     >Cics.CreateJet<
```

Creating the key store and WebSphere certificate

In order to enable signature and encryption processing, we need to create a certificate and key pair to be used by WebSphere. The certificate and key pair are stored in a key store.

Table 9-4 shows the data used to create the key store and the WebSphere certificate.

Table 9-4 Client key store and certificate values

Field	Value
Key store name	wasks
Filename	was.jks
Storepass	iiso
Storetype	JKS
Distinguished Name	CN=wascert, O=IBM, C=US
Key size	1024
Alias	wascert_rsa
Certificate file	wascert_rsa.cer

Both the WebSphere public and private keys are associated to the certificate.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000353

We performed the following tasks using the WebSphere Administration Console:

- Generating the client key store
- Creating the client certificate
- Extracting the client certificate file

Note: The capability to create certificates and key stores using the WebSphere admin console is new in WebSphere Application Server V6.1. If you are using an earlier version of WebSphere, you can use a key management tool such as *iKeyMan* or *keytool*. See "Importing the certificate" on page 277 for an example of using iKeyMan.

Creating the client keystore

Perform the following steps to create the client key store:

1. Select **Security → SSL certificate and key management**.
2. Under Related Items select **Key stores and certificates**.
3. The resulting panel shows you all the keystores that you have configured through the administrative console. Click **New** and enter these values:

Name:	`wasks`
Path:	`c:\SG247206\keystore\was.jks`
Password:	`itso`
Type:	`JKS`

4. Click **OK**.

Creating the client certificate

Use the following steps to create the personal certificate for the client key store:

- Select the **wasks** key store.
- Under Additional Properties click **Personal certificates**.
- Click **Create self-signed certificate** and enter these values:

Alias:	`wascert_rsa`
Key size:	`1024`
Common name:	`wascert`
Validity Period:	`365 (days)`
Organization:	`IBM`
Country:	`US`

- Click **OK**.

Be sure to save the configuration changes in the administrative console.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000354

The Java **keytool** command can be used to list the contents of a key store:

```
keytool -list -keystore was.jks -storepass itso -v
```

The keytool command comes with the Java runtime environment. You might need to add the path to your java jre using the command.

```
set PATH=<WAS_HOME>\java\jre\bin;%PATH%
```

In this command, <WAS_HOME> is the directory where your WebSphere is installed (usually C:\Program Files\IBM\WebSphere\AppServer).

Adding the CICS certificate to the WebSphere key store

To make the public key of the CICS certificate available to WebSphere, we performed the following steps:

1. We used the RACF command shown in Example 9-14 to extract the certificate from the RACF database to a sequential data set.

Example 9-14 RACF command to extract CICS certificate from RACF

```
RACDCERT ID(CICSUSR1) EXPORT (LABEL('CICSCERT')) DSN('CICS.CICSCERT.DER')
FORMAT(CERTDER)
```

The RACDCERT EXPORT command exports the certificate with label 'CICSCERT' to the MVS data set CICS.CICSCERT.DER as a DER encoded X.509 certificate.

2. We FTPed the CICS certificate to the WebSphere machine.

Example 9-15 shows how we used FTP to send the exported certificate (in binary format) to our WebSphere machine.

Example 9-15 FTP CICS certificate to WebSphere machine

```
C:\Documents and Settings\Administrator>ftp wtsc66.itso.ibm.com
Connected to wtsc66.itso.ibm.com.
220-FTPMVS1 IBM FTP CS V1R7 at wtsc66.itso.ibm.com, 21:50:57 on 2006-08-15
220 Connection will close if idle for more than 5 minutes.
User (wtsc66.itso.ibm.com:(none)): MIGFLM
331 Send password please.
Password:
230 MIGFLM is logged on.  Working directory is "MIGFLM.".
ftp> lcd c:\SG24\205\keystore
Local directory now C:\SG24\205\keystore.
ftp> bin
200 Representation type is Image
ftp> cd 'CICS'
250 "CICS." is the working directory name prefix.
ftp> get CICSCERT.CER
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000355

```
200 Port request OK.
125 Sending data set CINS.CICSCERT.DER
250 Transfer completed successfully.
ftp: 740 bytes received in 0.00Seconds 740000.00Kbytes/sec.
ftp> bye
221 Quit command received.
Goodbye.
```

3. We imported the CICS certificate into the WebSphere key store using the admin console.

   Figure 9-16 shows the certificate being imported using the admin console.

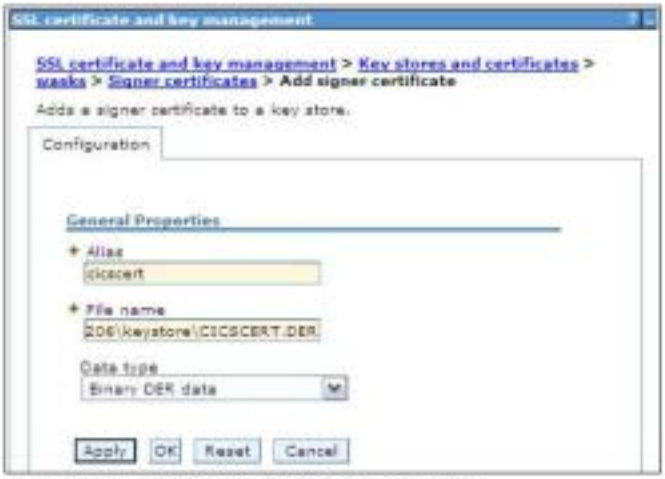


Figure 9-16 Import CICS certificate into key store

Adding the WebSphere certificate to RACF

Perform the following steps to make the public key of the WebSphere certificate available to CICS:

1. Extract the certificate from the key store using the WebSphere administration console.

   In "Creating the key store and WebSphere certificate" on page 324 we show how we created the WebSphere certificate. We now extract the certificate from the jks to file wascert.der (a Binary DER file).

   The certificate has both public and private keys. However, if you distribute the certificate, you must ensure that you only send the public key and not the private key.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 356/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000356

Follow these steps to export the public key from the wasks key store:

a. Locate the client personal certificates by selecting **SSL certificate and key management → Key stores and certificates → wasks → Personal certificates**.

b. Select the check box next to the newly created certificate `wascert_rsa`.

c. Click **Extract**.

d. For Certificate file name, specify `c:\SG247206\keystore\wascert.der`

e. For Data Type select **Binary DER data**.

f. Click **OK**.

Figure 9-17 shows the certificate being extracted using the admin console.



Figure 9-17 Extract client X.509 certificate from key store

2. FTP the WebSphere certificate to the host system.

   Before performing the FTP, we first allocated a sequential data set on the host machine SCTS66 to receive the certificate. This data set is allocated with a record format VB, record length 84 and block size 27998.

   We then used FTP as shown in Example 9-16 to send the exported certificate (in binary format) to our host machine.

Example 9-16 FTP client X.509 certificate to host

```
C:\Documents and Settings\Resident>ftp wtsc66.itso.ibm.com
Connected to wtsc66.itso.ibm.com.
220-FTPMVS1 IBM FTP CS V1R7 at wtsc66.itso.ibm.com, 21:04:02 on 2006-08-11.
220 Connection will close if idle for more than 5 minutes.
User (wtsc66.itso.ibm.com:(none)): NIGELW
331 Send password please.
Password:
```


Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 357/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000357

```
230 MIGELW is logged on.  Working directory is "MIGELW.".
ftp> lcd C:\SG247206\keystore
Local directory now C:\SG247206\keystore.
ftp> cd 'CICS'
250 "CICS." is the working directory name prefix.
ftp> bin
200 Representation type is Image
ftp> put wascert.der
200 Port request OK.
125 Storing data set CICS.WASCERT.DER
250 Transfer completed successfully.
ftp: 580 bytes sent in 0.00Seconds 580000.00Kbytes/sec.
ftp> bye
221 Quit command received.
Goodbye.
```

3. Import the client certificate into RACF and connect it to the CICS key ring.

   We used the RACF commands shown in Example 9-17.

Example 9-17 RACF commands to import WebSphere certificate into RACF

```
RACDCERT ID(WEBAS1) ADD('CICS.WASCERT.DER') WITHLABEL('WASCERT') TRUST
RACDCERT ID(CICSUSR) CONNECT(ID(WEBAS1) LABEL('WASCert') RING(Cics.Cicsdc1))
```

   - The RACDCERT ADD command adds the client certificate with label 'WASCERT' as a trusted certificate.
   - The RACDCERT CONNECT command connects the client certificate to the CICS key ring. ID(WEBAS1) indicates that the certificate being added to the key ring is a user certificate, and WEBAS1 is the user ID that is associated with this certificate.

   The certificate can be listed using this command:

```
RACDCERT ID(WEBAS1) LIST
```

   Example 9-18 shows the output of the RACDCERT LIST command.

Example 9-18 Listing of client X.509 certificate (WASCERT)

```
Digital certificate information for user WEBAS1:

  Label: WASCERT
  Certificate ID: 2QbuuC.B4xHxwLDado]
  Status: TRUST
  Start Date: 2006/08/01 14:36:07
  End Date:   2007/08/01 14:36:07
  Serial Number:
       >4A3A3C<
  Issuer's Name:
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000358

```
      "CN=wascert,OU=,O=IBM,L=,SP=,,C=US"
Subject's Name:
      "CN=wascert,OU=,O=IBM,L=,SP=,,C=US"
Private Key Type: None
Ring Associations:
  Ring Owner: CICSSD
  Ring:
     "Ciws.Ciwss3cl"
```

9.5 Signing a SOAP message

To provide integrity on the client's request message, we can add an XML digital signature to the message. CICS supports digital signatures in both service provider and service requester modes.

In this section we show how a SOAP request message can be signed by WebSphere Application Server and how the signature can be verified by CICS and how the response message can be signed by CICS and verified by WebSphere.

- The SOAP request message contains an X.509 certificate (the WebSphere certificate) that is used to sign the body of the message.
- The SOAP response message also contains an X.509 certificate (the CICS certificate) that is used to sign the body of the message.

The X.509 certificates are transported within a *BinarySecurityToken* element.

We also show how the WebSphere certificate can be used to authenticate the service requester. DFHWSSE1 verifies the signature and uses the certificate to determine the user ID under which the ORDR transaction will be run. This scenario is shown in Figure 9-18.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000359

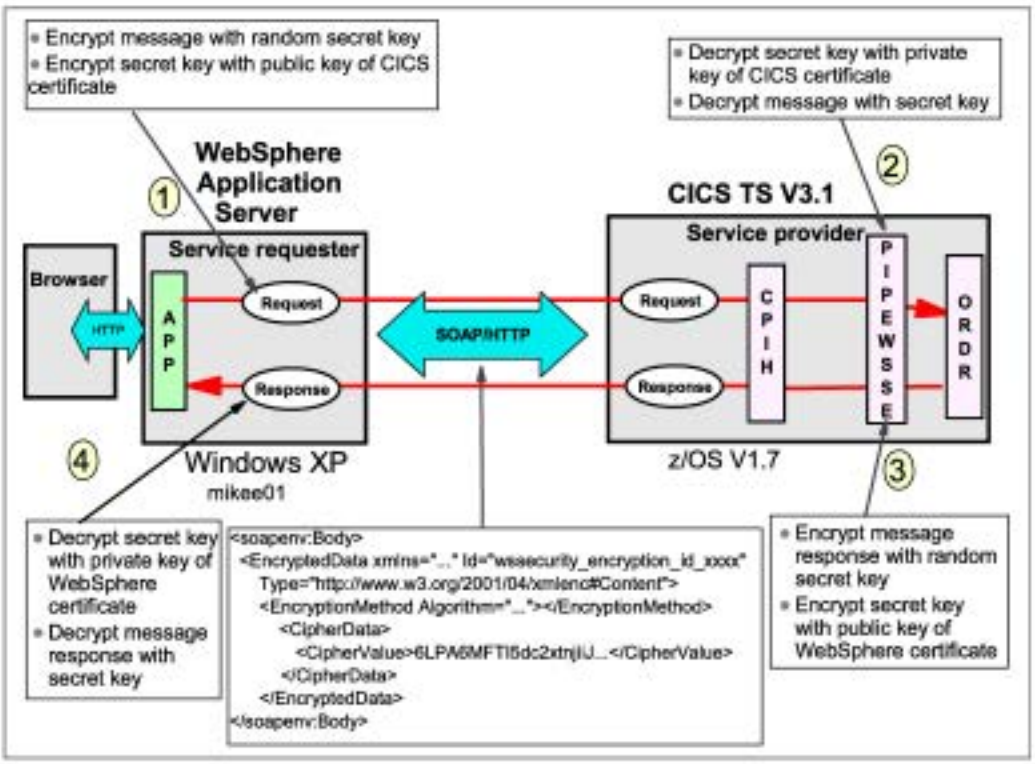


Figure 9-18 Enabling CICS WS-Security support for signatures

1. The service requester (WebSphere in our example) generates a BinarySecurityToken element from the WebSphere X.509 certificate. It then signs the message with its private key so the service provider (CICS in our example) knows that the message can only be sent by the WebSphere application server.
2. CICS validates the signature with the WebSphere public key. It also uses the owning user ID of the WebSphere X.509 certificate (WEBAS1) to run the ORDR transaction for the placeOrder service.
3. CICS then signs the response message with its private key so that WebSphere knows that the message can only be sent by the CICS region.
4. WebSphere validates the signature with the public key of CICS.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 360/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000360

Important: There is a significant performance overhead when using WS-Security and XML digital signatures. See 7.5, "Comparison of transport versus SOAP message security" on page 241 for recommendations on choosing between WS-Security and SSL when implementing an integrity security solution.

We document the following procedures to enable support for XML signature processing:

- Configuring the service requester for signature processing
- Configuring CICS for signature processing
- Testing the signature test scenario

Note: In our signature test scenario we sign both the request and response messages. It is also possible to sign only the request, or to sign only the response.

9.5.1 Configuring the service requester for signature processing

In this section we show how to configure the client J2EE application to send and receive signed SOAP messages. At a high level, the steps to do this are:

- Import the CatalogSec2 WebSphere Application to the Application Server Toolkit (AST).
- Configure the request generator for signing.
- Configure the response consumer for signing.
- Re-deploy the Web Service client application.

In the following sub-sections we provide step-by-step details for each procedure.

Importing the base application

To configure our Web Service client to use WS-Security, we used the IBM WebSphere Application Server Toolkit V6.1 (AST).

See "Importing the base application" on page 305 for instructions on how to import the application. For the signature test scenario, we specified:

- The name of the workspace folder as `C:\AST\CatalogSec2\CatalogSec2Signing`
- The name of the EAR project as `CatalogSec2Sign`.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000361

Configuring the request generator for signing

Perform the following steps to configure the request generator for signing:

1. Expand the **Dynamic Web Project** (**CatalogSec2Web**) in the Project Explorer and open (double-click) the deployment descriptor as shown in Figure 9-19.

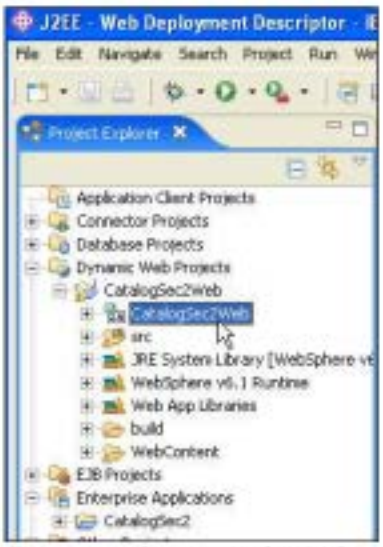


Figure 9-19 Opening the CatalogSec2Web Deployment Descriptor

2. Go to the WS Extension page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Request Generator Configuration**.
   c. Expand **Integrity**, click **Add**, and in the Integrity dialog box (Figure 9-20) enter the following data:
      - Enter a name identifying the part, for example, `int_body`.
      - Select the order in which the signature is generated. Multiple integrity parts can be specified. In our case, we selected 1.
      - Click **Add** for the Message Parts, and one integrity part is created. The default created part is the SOAP body. We accepted the default.
   d. Click **OK** and save the configuration by pressing Ctrl+s.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 362/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000362

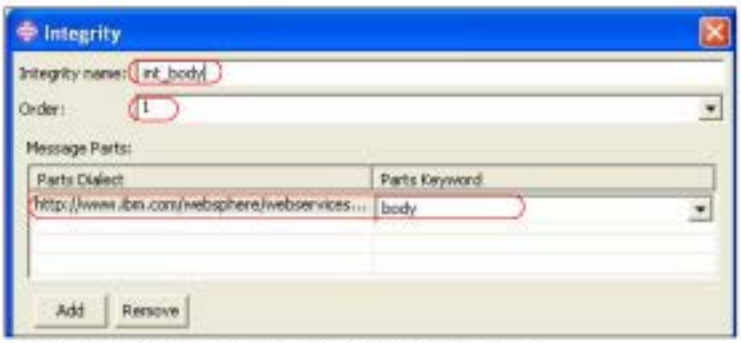


Figure 9-20 Integrity dialog in the WS Extensions page

3. Open the WS Binding page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Security Request Generator Binding Configuration**.
   c. Expand **Token Generator** and click **Add.** In the Token Generator dialog box (Figure 9-21) enter the following data:
      - Enter a Token generator name, for example, `gen_dsigtgen`.
      - For the Token generator class, select the **X509TokenGenerator**.
      - Do not select a Security Token.
      - Select **Use value type**, and then select **X509 certificate token v3** and the X509CallbackHandler.
      - Select **Use key store** and make the appropriate entries. Ours were:

        Password: `itso`
        Path: `C:\SG247206\keystore\was.jks`
        Type: `JKS`

      - Click **Add** under Key and make the appropriate entries. Ours were:

        Alias: `wascert_rsa`
        Key password: `itso`
        Key name: `CN=wascert, O=IBM, C=US`

        Note: This is the key name of the WebSphere certificate whose private key is used to sign the request message.

      - Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 363/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000363

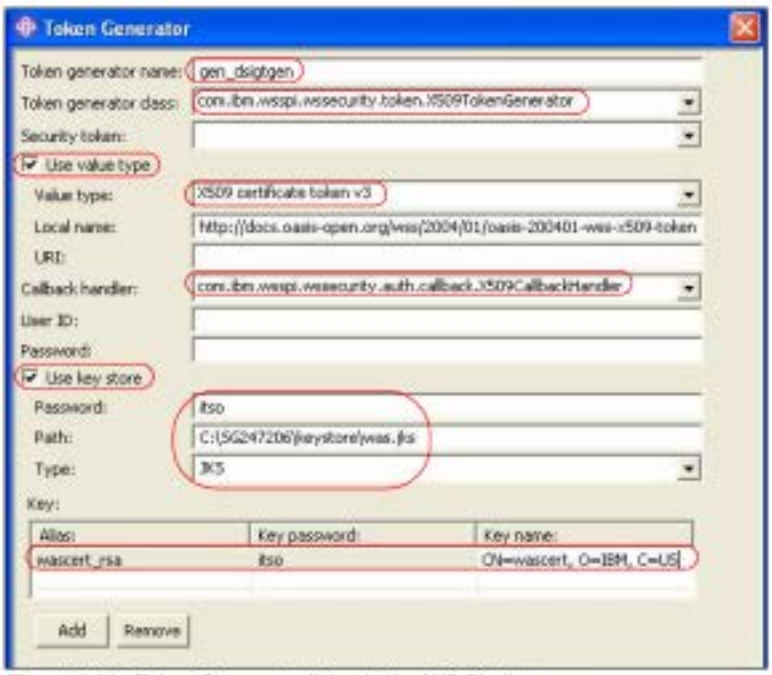


Figure 9-21 Token Generator dialog in the WS Binding page

4. Expand **Key Locators**, click **Add**, and in the Key Locator dialog box (Figure 9-22) enter the following data:

   a. Enter a Key locator name, for example, `gen_dsigklocator`.

   b. Select **KeyStoreKeyLocator** as the Key locator class.

   c. Select **Use key store** and make the appropriate entries. Ours were:

      Password: `itso`
      Path: `C:\SG247206\keystore\was.jks`
      Type: `JKS`

   d. Click **Add** under Key and make the appropriate entries. Ours were:

      Alias: `wascert_rsa`
      Key password: `itso`
      Key name: `CN=wascert, O=IBM, C=US`

   e. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
Online Notary.net
Page 364/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000364

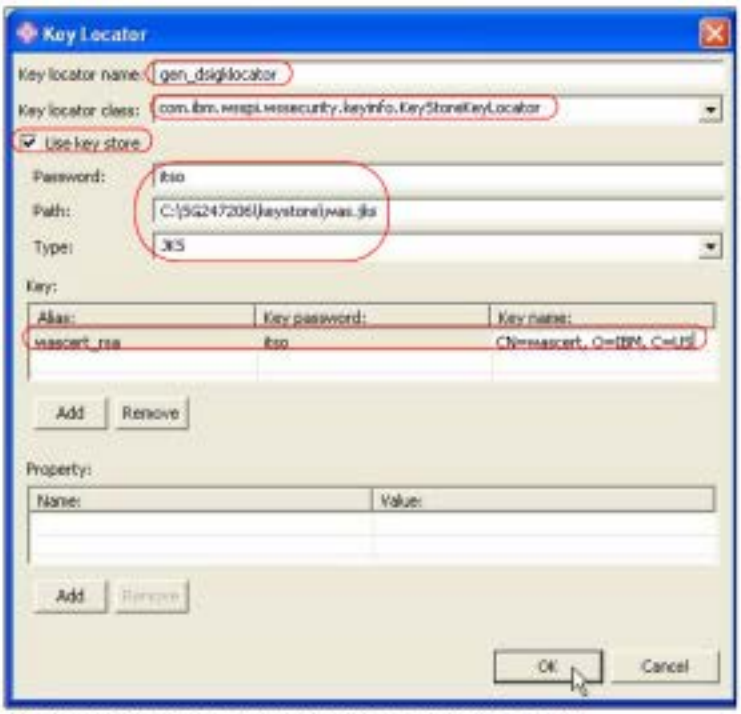


Figure 9-22 The Key Locator dialog in the WS Binding page

5. Expand **Key Information**, click **Add**, and in the Key Information dialog box (Figure 9-23) enter the following data:
   a. Enter a Key information name, for example, `gen_dsigkeyinfo`.
   b. Select **STRREF** (Direct reference) as the Key information type. The Key information class is filled automatically.

      The possible Key information types are:

      - STRREF Direct reference
      - EMB Embedded reference
      - KEYID Key identifier reference
      - KEYNAME Key name reference
      - X509ISSUER x.509 issuer and serial number reference

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 365/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000365

Note: We specify STRREF because we want WebSphere to send the public key as part of the complete certificate, since we will use the certificate for authentication in addition to decrypting the signed message.

It is also possible to configure an indirect reference to the public key, such as the key identifier or the key name. In this case, CICS uses the key reference to identify the RACF certificate that contains the public key.

c. Select **Use key locator** and select **gen_dsigklocator** from the Key locator drop-down list. Select the predefined Key name **CN=wascert, O=IBM, C=US**.

d. Select **Use token**, and select **gen_dsigtgen** from the drop-down list.

e. Click **OK**.

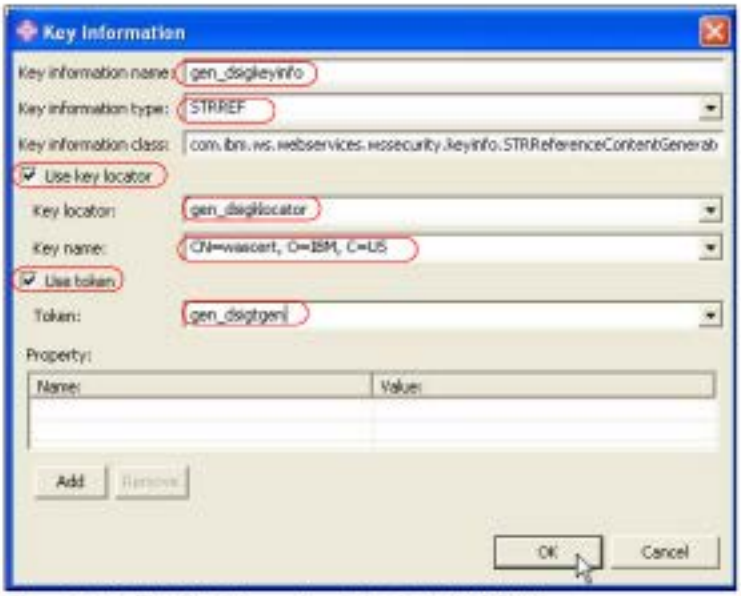


Figure 9-23 The Key Information dialog in the WS Binding page

6. Expand **Signing Information**, click **Add**, and in the Signing Information dialog box (Figure 9-24) enter the following data:

   a. Enter a Signing Information Name, for example, `sign_body`.

   b. Enter a Key information name, for example, `sign_kinfo`.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 366/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000366

c. Select the Key information element previously defined. In our case this was **gen_dsigkeyinfo**.

d. Click **OK**.

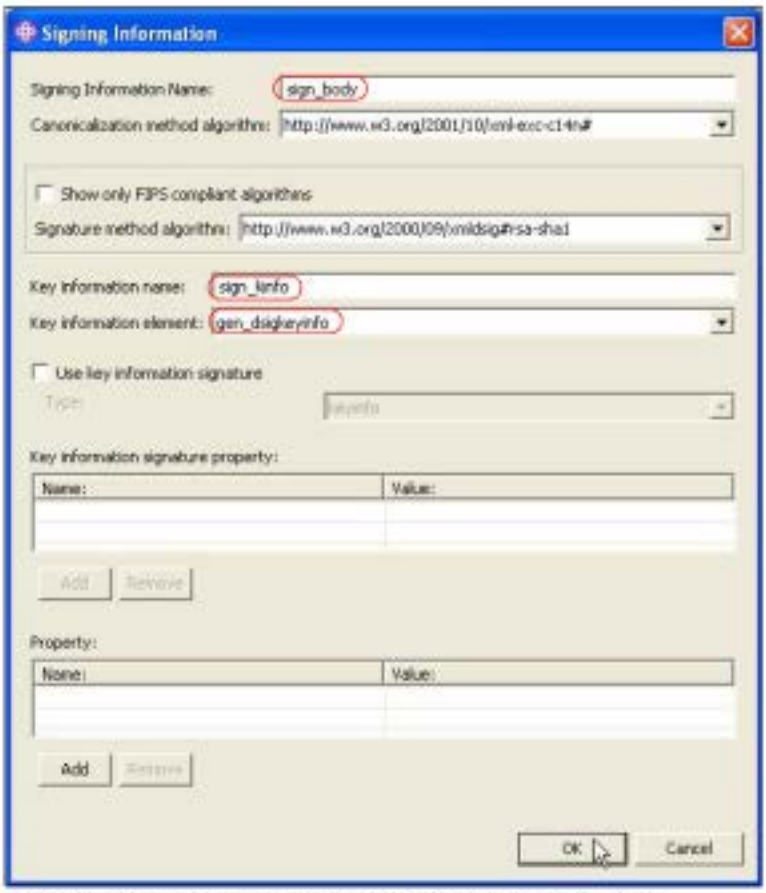


Figure 9-24 The Signing Information dialog in the WS Binding page

7. Expand **Part References**, click **Add**, and in the Part Reference dialog box (Figure 9-25) enter the following data:

   a. Enter a Part reference name, for example, `sign_part`.

   b. Select the Integrity part from the list of parts defined on the WS Extensions page. In our case, we select **int_body**.

   c. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 367/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000367



Figure 9-25 The Part Reference dialog in the WS Binding page

8. Expand **Transforms**, click **Add**, and in the Transform dialog box (Figure 9-26) enter the following data:
   a. Enter a name, for example, `sign_trans`.
   b. Click **OK**.



Figure 9-26 The Transform dialog in the WS Binding page

9. Save the configuration by pressing Ctrl+s.

Configure the response consumer for signing

In this section we show how to configure the client J2EE application in order to receive a signed SOAP message.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000368

Use the following steps to do this:

1. Expand the **Dynamic Web Project** (**CatalogSec2Web**) in the Project Explorer and open (double-click) the deployment descriptor.
2. Go to the WS Extension page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Response Consumer Configuration**.
   c. Expand **Required Integrity** and click **Add**. In the Required Integrity dialog box (Figure 9-27) enter the following data:
      - Enter a Required integrity name, for example, `reqint_body`.
      - Select the Usage type, either Required or Optional. If the usage type is Required, an unsigned response message throws a SOAP fault. If the usage type is Optional, an unsigned message is received. We selected **Required**.
      - Click **Add** for Message Parts, and one message part is created. The default created part is the SOAP body. We accepted the default.
   d. Click **OK** and save the configuration by pressing Ctrl+s.

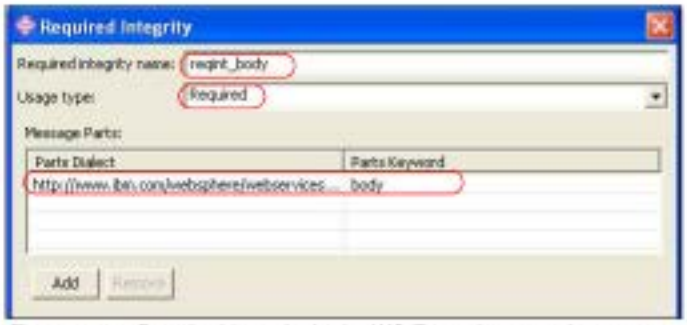


Figure 9-27 Required integrity in the WS Extension page for response consumer

3. Open the WS Binding page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Security Response Consumer Binding Configuration**.
   c. Expand **Token Consumer**, click **Add**, and in the Token Consumer dialog box (Figure 9-28) enter the following data:
      - Enter a Token consumer name, for example, `con_dsigtcon`.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000369

- Select **com.ibm.wsspi.wssecurity.token.X509TokenConsumer** as the Token consumer class.
- Select **Use value type**.
- Select **X509 certificate token**. The Local name is selected automatically.

> **Note:** It is important to specify X509 certificate token for the token value type. An error occurred in our test when we specified X509 certificate token v3.

- Select **Use jaas.config** and enter `system.wssecurity.X509BST` as jaas.config name.
- Select **Use certificate path settings**.
- Select **Trust any certificate**.

> **Important:** We specified **Trust any certificate** for our test. By selecting this option, the signature is validated by the certificate sent with the message without the certificate itself being validated.
>
> However, in a production environment you are unlikely to use this option because it allows any client with a valid XML digital signature certificate to have access to your WebSphere application server.
>
> WebSphere provides different ways of specifying trust for digital signature certificates.
>
> - If you use a self-signed certificate, you can specify the issuer name or serial number of trusted certificates in a jaas.config property.
> - For both self-signed and CA-signed certificates, you can specify a *trust anchor* and a *trusted certificate store* using a Certificate path reference.

d. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Page 370/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000370

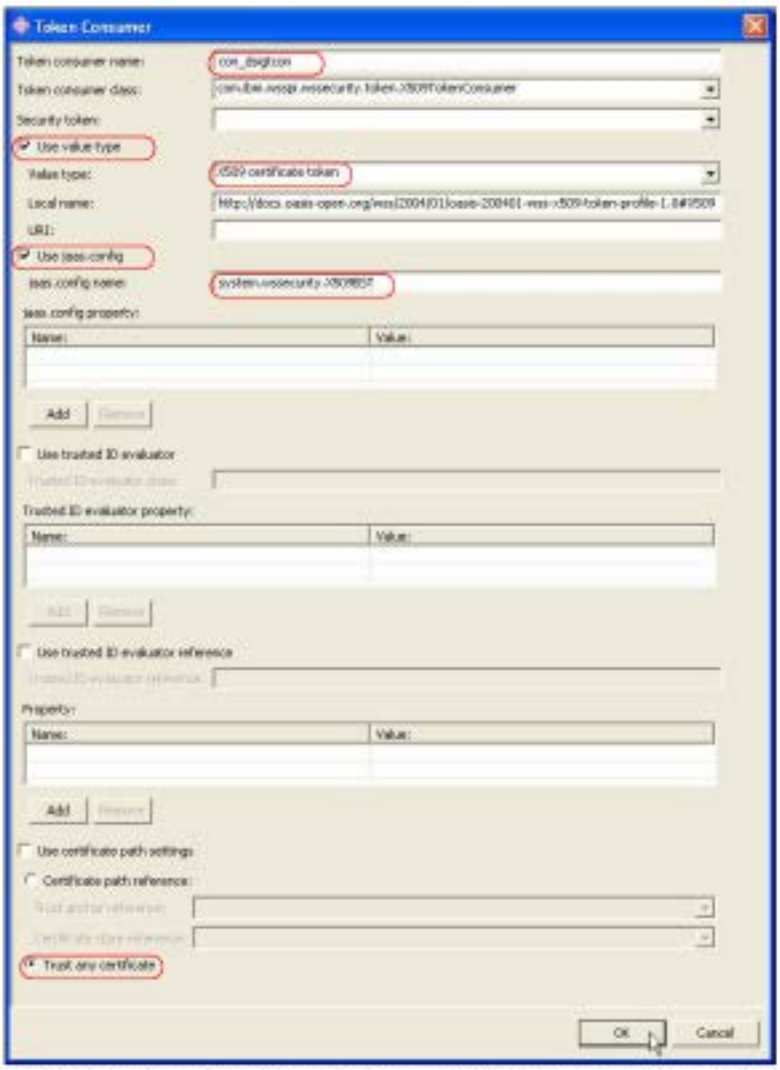


Figure 9-28 Token Consumer dialog in the WS Binding page for Response Consumer

4. Expand **Key Locators**, click **Add**, and in the Key Locator dialog box (Figure 9-29) enter the following data:
   a. Enter a Key locator name, for example, `con_dsigklocator`.
   b. Select **KeyStoreKeyLocator** as the Key locator class.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 371/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000371

c. Select Use key store and make the appropriate entries. Ours were:

Password:	itso
Path:	C:\SG247206\keystore\was.jks
Type:	JKS

d. Click **Add** under Key and make the appropriate entries. Ours were:

Alias:	cicscert
Key password:	itso
Key name:	CN=CICSCERT, O=ITSO, C=US

> **Note:** This is the key name of the CICS certificate. The private key of this certificate is used to sign the response message.

e. Click **OK**.

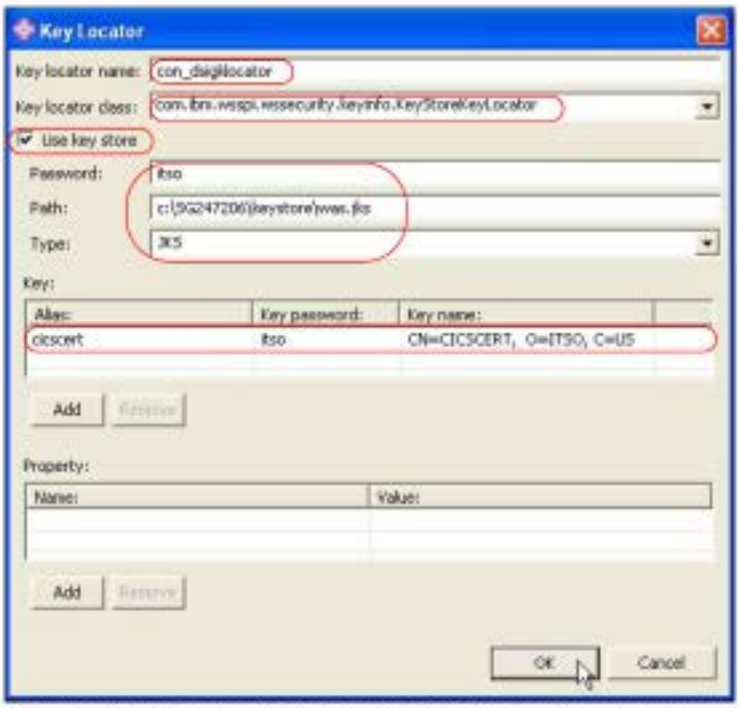


Figure 9-29 Key Locator dialog in the WS Binding page for response consumer

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 372/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000372

5. Expand **Key Information**, click **Add**, and in the Key Information dialog box (Figure 9-30) enter the following data:

   a. Enter a Key information name, for example, `con_dsigkeyinfo`.

   b. Select **STRREF** as the Key information type. The Key information class is filled automatically.

   > **Note:** We specify STRREF because CICS always sends the complete certificate when signing an outbound message; therefore, the WebSphere response consumer must be configured to expect the CICS certificate in the security header of the response message.

   c. Select **Use key locator** and select **con_dsigklocator** from the Key locator drop-down list. Select the predefined Key name as **CN=CICSCERT, O=ITSO, C=US**.

   d. Select **Use token** and select **con_dsigtcon** from the Token drop-down list.

   e. Click **OK**.

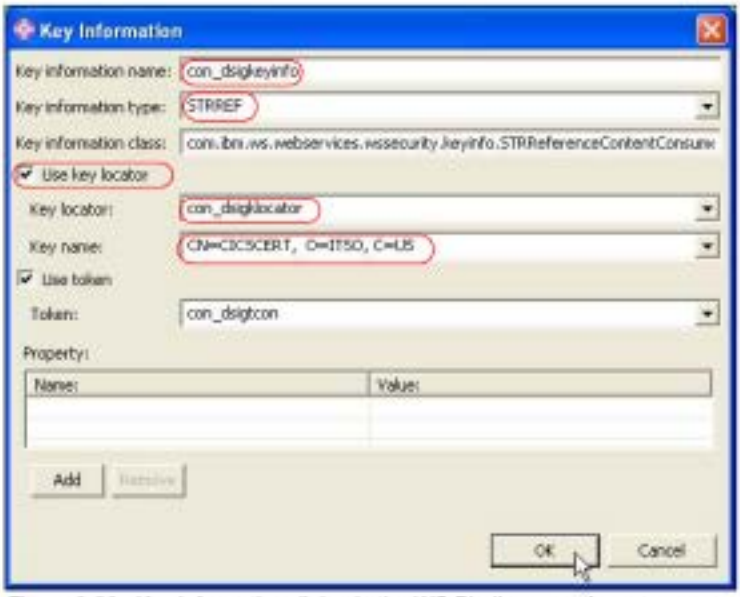


Figure 9-30 Key Information dialog in the WS Binding page for response consumer

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 373/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000373

6. Expand **Signing Information**, click **Add**, and in the Signing Information dialog box (Figure 9-31) enter the following data:
   a. Enter a Signing Information Name, for example, `sign_body`.
   b. Select **http://www.w3.org/2001/10/xml-exc-c14n#** as the Canonicalization method algorithm.
   c. Select **http://www.w3.org/2000/09/xmldsig#rsa-sha1** as the Signature method algorithm.
   d. Click **Add** in Signing key information.
      - Enter a Key information name, for example, `sign_kinfo`.
      - Select the **con_dsigkeyinfo** from the Key information element drop-down list.
   e. Click **OK**.

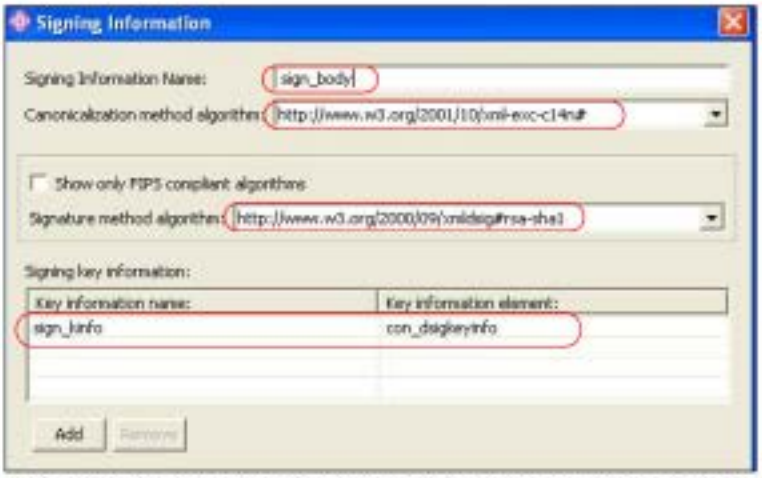


Figure 9-31 Signing Information dialog in the WS Binding page for response consumer

7. Expand **Part References**, click **Add**, and in the Part Reference dialog box (Figure 9-32) enter the following data:
   a. Enter a Part reference Name, for example, `sign_part`.
   b. Select **reqint_body** as the Required integrity.
   c. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 374/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000374



Figure 9-32 The Part Reference dialog in the WS Binding page for response consumer

8. Expand **Transforms**, click **Add**, and in the Transform dialog box (Figure 9-33) enter the following data:
   a. Enter a Name, for example, `sign_trans`.
   b. Select **http://www.w3.org/2001/10/xml-exc-c14n#** as the algorithm.
   c. Click **OK**.



Figure 9-33 The Transform dialog in the WS Binding page for response consumer

9. Save the configuration by pressing Ctrl+s.

Redeploying the Web service application

After configuring a signature security constraint for the service requester application, we exported a new EAR file called CatalogSec2_WS-Security_Signature.ear. We followed the same process that is

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 375/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000375

described in "Installing the service requester" on page 87 to deploy the CatalogSec2 WS-Security Signature.ear.

For further information on configuring WS-Security in WebSphere refer to *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257.

9.5.2 Configuring CICS for signature processing

In this section we show how to configure the CICS pipeline to receive and send signed SOAP messages, and to run the placeOrder request under the user ID associated with the WebSphere certificate contained in the request message.

Example 9-19 shows the pipeline configuration file that we used for the signature test scenario. It includes the message handler DFHWSSE1, and the configuration information for the handler.

Example 9-19 Pipeline config file, ITSO_7206_wssec_signature_provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/hts/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/hts/cics/pipeline
  provider.xsd ">
    <transport>
      <default_transport_handler_list>
          <handler>
             <program>CIKSMSGH</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
    </transport>
    <service>
      <service_handler_list>
        <handler>
          <program>SMI====6</program>
          <handler_parameter_list/>
        </handler>
        <wsse_handler>
          <dfhwsse_configuration version="1">
            <authentication mode="signature">
            <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
            </authentication>
            <expect_signed_body/>
            <sign_body>
              <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
              <certificate_label>CICSCERT</certificate_label>
            </sign_body>
          </dfhwsse_configuration>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000376

```
        </wsse_handler>
      </service_handler_list>
      </terminal_handler>
        <cics_soap_1.2_handler/>
      </terminal_handler>
    </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

The <wsse_handler> element shown in Example 9-19 contains a <dfhwsse_configuration> element that specifies configuration information for DFHWSSE1.

Important: If the <wsse-handler> element is included in a provider pipeline configuration, CICS processes signed SOAP messages by default.

- The <authentication mode="signature"> element specifies that inbound messages must contain an X.509 certificate in a BinarySecurityToken.

  If <authentication mode="signature"> is specified then the client X.509 certificate must be imported to RACF and attached to the CICS key ring since CICS runs the service request under the user ID associated with the client certificate.

  Note: CICS does not support certificate name filtering for signed SOAP messages.

  If <authentication mode="signature"> is not specified, and if the complete X.509 certificate is included in the request message, then the client X.509 certificate does not need to be imported to RACF since the public key required to validate the signature is extracted from the X.509 certificate contained in the SOAP message.

- <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm> is a child element of the *<authentication>* element that specifies the URI of the signature algorithm.

  Note: We found that it is necessary to specify a valid signature algorithm in service provider mode even if the algorithm used in the decryption of the signature is the one specified in the SOAP message rather than in the pipeline configuration file.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 377/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000377

- The <expect_signed_body/> element indicates that the *<body>* of the inbound message must be signed. If the body of an inbound message is not correctly signed, CICS rejects the message with a security fault.

  We specify <expect_signed_body/> because we want to prevent the placeOrder service being accessed unless the body of the SOAP message is signed.
- The <sign_body/> element indicates that the *<body>* of the outbound message must be signed, and provides information about how the message is to be signed.
- <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm> is a child element of the *<sign-body>* element which specifies the URI of the signature algorithm to be used for signing the response message. This is the only signature algorithm supported by CICS for outbound messages.

  Attention: During our tests we encountered codepage issues when specifying the *<algorithm>* element. The configuration file should be coded in an English encoding such as EBCDIC-CP-US. When using a terminal emulation session with a non-English codepage, we noticed that the symbol '#' was a different hex value in the non-English codepage than in the EBCDIC-CP-US codepage (in which the symbol '#' is x'7B'). This resulted in error DFHPI0723 with the message:

  ```
  'The value for the algorithm specified for the authentication is
  not supported'.
  ```

  We recommend that you use a terminal emulation session with an English codepage when editing the pipeline configuration file.
- <certificate_label>CICSCERT</certificate_label> is a child element of the <sign-body> element that specifies the label associated with the CICS certificate installed in RACF. This certificate contains the PKDS key label of the private key that is used to sign the message response. The public key associated with the private key is then sent in the SOAP response message, allowing the signature to be validated by WebSphere.

We completed the CICS configuration by changing the pipeline PIPEWSSE (used for the placeOrder service) to specify the pipeline configuration file that we have created for signature processing. We used the following CEDA command:

```
CEDA ALTER PIPELINE(PIPEWSSE) GROUP(S3C1EXWS)
Configfile(/CIWS/S3C1/config/ITSO_7206_wssec_signature_provider.xml)
```

We then re-installed the changed pipeline resource definition.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000378

9.5.3 Testing the signature scenario

An example of the signed request message sent by WebSphere is shown in Example 9-20.

Example 9-20 Signed SOAP request message

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse
      curity-secext-1.0.xsd">
      <wsse:BinarySecurityToken
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-s
        oap-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="X509Token_1"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse
        curity-utility-1.0.xsd">MIIJCQUCCAmgAwIBAgIEVbOdzDANBgkqhkiG9w0BAQUFADs
        ZNQswCQYDVQQGEwJVUzEUMBIGA1UEChMLRVF1aWZheCBTZWN1cmUxJjAkBgNVBAsTHUVx
        dWlmYXggU2VjdXJlIENlcnRpZmljYXRlIEF1dGhvcml0eTANBgkqhkiG9w0BAQUFAAOBgQ
        CSgYEAwkTaDEMMAoGA1UECxMDQ0RTMRswGQYDVQQDExJDeWJlcnRydXN0IEdsb2JhbCBS
        b290MB4XDTA0MDUxMzE4MjM0M1oXDTA5MDUxMzE4MjM0M1owRDELMAkGA1UEBhMCVVMx
        FjAUBgNVBAoTDUVudHJ1c3QubmV0MSIwIAYDVQQLExlFbnRydXN0IFB1YmxpYyBDZXJ0
        aWZpY2F0aW9uIEF1dGhvcml0eTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAwqZB
        aMoGQpzybJ+P3vCwNRbLVK3TTbrduxQ7YuK1HVUCey2VVSebES2yDaTrCLJ6/E2YTpB7
        HEt6+jbczvdF6xt/drfs7JD1ZVHX9zXDQHkm1EcovSKGr1w4nbyDkhI4ZJ2dES1XjnY
        d1T2ouE2BcRMTveJqfMC3mkn+jcn0EcSZlFrCo1fVN2/WL1axQwSJDm6coJFCNL4VY2V
        AAsjFI4PHfi9A1LciGcQ4SAwIYPhuSs7ASIANRcqHk+G9w9B4QJHAQUHgUEPLSFGGVW4rua
        zRDhIcdXFHHA4b6OrzIwObn+nh7Jd/uRIIyYHh6fcw+k4b+9F9U/IIPzIbJzgnpcM/rqHKK
        PIcb/vIB/HUuX+qrcGN+c12XBfrLs1S2IUPU66+qQx/JA3ygSJCuYfzKr2CcCPLFQ+cMMk
        nuj+UUIcAoUp+GF4Q==
      </wsse:BinarySecurityToken>
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-ex
            c-c14n#">
            <ec:InclusiveNamespaces PrefixList="xsi xsd soapenv soapenc wsse
              ds" xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" />
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod Algorithm=
            "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          <ds:Reference URI="#wssecurity_signature_1c_0">
            <ds:Transforms>
              <ds:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                <ec:InclusiveNamespaces PrefixList="xsi xsd soapenv soapenc wsu
                  p036" xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" />
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000379

```
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm=
            "http://www.w3.org/2000/09/xmldsig#sha1" />
          <ds:DigestValue>fh93ysjsoaBCwZ9RQqH2mI+WdYQ=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>I53WN8zcbfRmnz3LwVf3vZeFFDppZWAkgBU9+Sfu/I7J63WB3LsN
        3fRngwt7qLaRM3qGzjpsMNCbrKssyOP+dr5yBwvUIz4FukxyJ5pASVot7qPRRfj9p60/Y
        01u26zON96Q5cljkqWpl6+tNZqHwLiNfpwQyJggII+8izQU8nkQ=
      </ds:SignatureValue>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:Reference URI="#x509bst1 1"
            ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-ws
            s-x509-token-profile-1.0#X509v3" />
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
    </ds:Signature>
  </wsse:Security>
</soapenv:Header>
<soapenv:Body wsu:Id="wssecurity_signature_id_0"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
  ty-utility-1.0.xsd">
  <p635:DFHQXCVM xmlns:p635="urn://www.DFHQXCVM.DFHQXCP5.Request.com">
    <p63b:ca_request_id>ECIOR3R</p63b:ca_request_id>
    <p63b:ca_return_code>0</p63b:ca_return_code>
    <p63b:ca_response_message>CWBQZM1Z</p63b:ca_response_message>
    <p635:ca_order_request>
      <p63b:ca_userid>USER</p63b:ca_userid>
      <p635:ca_charge_dept>ITSO</p635:ca_charge_dept>
      <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
      <p635:ca_quantity_req>1</p635:ca_quantity_req>
      <p635:filler1 xsi:nil="true" />
    </p635:ca_order_request>
  </p635:DFHQXCVM>
</soapenv:Body>
</soapenv:Envelope>
```

- The header includes the mustUnderstand="1" attribute, which indicates that either this header must be processed or a SOAP fault thrown.
- The <wsse:BinarySecurityToken> contains the base64binary encoding of the WebSphere certificate. This includes the public key that CICS uses to verify the signature.
- The <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" /> element specifies the algorithm used to sign the message digest

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000380

- The <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" /> element specifies the algorithm used to produce the message digest.
- The <ds:DigestValue> element contains the value of the message digest.
- The <ds:SignatureValue> element contains the value of the signed message digest. It is the digest value encrypted with the requester's private key.

Figure 9-34 shows the ORDR transaction running with the user ID WEBAS1 (the user ID associated with the WebSphere certificate), while the CPIH transaction continues to execute with the CICSNW user ID.

```
 INQUIRE TASK
 STATUS:  RESULTS - OVERTYPE TO MODIFY
  Tas(CCCCCE5) Tra(CPIH)          Sus Tas Pri( 001 )
     Sta( ) Use(CICSNW  ) Ubn(B73C2E9E2114284E) Hty(RZCBNOT2)
  Tas(CCCCCE6) Tra(ORDR)          Sus Tas Pri( 001 )
     Sta( ) Use(WEBAS1  ) Ubn(B73C2E9E24801B9B) Hty(ECI     )
  Tas(CCCCCE8) Tra(CEMT) Fac(E024) Sus Ter Pri( 000 )
     Sta(SD) Use(NIGELW  ) Ubn(B73C2E9E2B044149) Hty(ZCIOWAIT)
  Tas(CCCCCE9) Tra(CEMT) Fac(E024) Sus Ter Pri( 255 )
     Sta(TO) Use(NIGELW  ) Ubn(B-3C25A641E9CCCE1
                                                    SrSID=SIDJ APPLID=A6P0S3CL
```

Figure 9-34 ORDR executing with user ID associated with WebSphere certificate

The corresponding signed response message sent by CICS is shown in Example 9-21.

Example 9-21 Signed SOAP response message

```
<?xml version="1.0" encoding="UTF8" standalone="no" ?>
 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Header>
     <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
        200401-wss-wssecurity-secext-1.0.xsd" SOAP-ENV:mustUnderstand="1"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
       xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
       wssecurity-utility-1.0.xsd"
       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
       <wsse:BinarySecurityToken
         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
         -soap-message-security-1.0#Base64Binary"
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000381

```
  ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x5
  09-token-profile-1.0#X509"
  wsu:Id="x509cert-CC0"MIIC4DCCAkigAwIBAgIEAOAVBjAKBggqhkjOPQQDBAUA
    QYDVQQGEwJVSzEPMA0GA1UECBMGTWVYIFFlcsgxETAPBgNVBAcTCEVYCRETDTELMAkG
    CwYDVQQKEwRJVFNMMQ0wCwYDVQQLEwRNQUNFMRIwEAYDVQQDEwlNQUNFIFFuQCBFeCAx
    FVREaMBYDVQQDEw9DSUNTQ3VSVDAeFw0wNDA4wNDBaFw0xNTAxMDEwMzU5WjBT
    IaMBgxCzAJBgNVBAcTAlVIMREwDwYDVQQIEwhOb3gwMJBSSzERMA8GA1UEBxMIRUBF
    UMPVFQxDTALBgNVBAoTBElJU09xDTALBgNVBAsTBFBTJDYxEjAQBgNVBAMTCUCBRElJg
    YVGqYElLIELxc1A/sqNVBMIIBMjCBlwYJKoZIhvcNAQMBMIGJAoGBAOLAAAGMAJC
    BiQKUgCCsEP+avGQtIkvbPJ+qNaUFF/9thk2aAJLr3Pup4KJvcA/JueLtn6sLQyQJVF
    snZgwQIIk97ELCCs755jwYdZcGSR2TgHxswdGrt4rt8EPgwcM3mSI6C9uWrn+Tvq1Jz
    NCDXpivdcRCTwRcQg+NUAJDKJAvdSr+bCwGi64LSdn2pQIDAQABuJIwcDARJ6g1g+kgB
    hvhCAQQEMHPwRZ4uZ2VuZGVkIGJ5IHRoZSBTZWNJdnIJe5BTZXJ2ZXIgZnQgZm4uIN
    gKFJRQQYpPA4GAJL+JwEB/wQE4wIEUDAdRghWMD4EFgQJ62++0kFhSysr93LQ62k5n
    yYaPgwRCYJKn7I+xcN4QFFRJ4NgYFATtVWjr2p7+471F37RaTpUne7xq7eMn7.QQ5ID
    u37ya+qC AxbIP574+Kgqn+egYyYXRY14mCsrpRq3H4R4/nFIvm1FUNprnCe28F+1Iq
    3p3rzn+hfuRY4+Ia4t4CK9rUPIIjsvgxpuIrWfcM+/JrxYXcY3U1h4L/RBBs/CQ17xJ

</wsse:BinarySecurityToken>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
      <c14n:InclusiveNamespaces
        xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"
        PrefixList="ds wsu xenc SOAP-ENV soapenv soapenv xsd xsi "/>
    </ds:CanonicalizationMethod>
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#TheBody">
      <ds:Transforms>
        <ds:Transform
          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
          <c14n:InclusiveNamespaces
            xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"
            PrefixList="wsu SOAP-ENV soapenv soapenv xsd xsi "/>
        </ds:Transform>
      </ds:Transforms>
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>sLbsd15PsgrzG25bDQdtyBoHDW0=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
```


Document Id: [illegible]
[illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000382

```
          <ds:SignatureValue>ZyxxxQ/n7yDaGDYwsIIS2bMIbOdMWNruFJ/tT5HuWi0Ob4N7k5
            DFccH27wQbluEVFjkMjkKzOnOLHNgIzGqaBU4cV3hu6VDKr4QgBCmxtO6yDpyOYYC/e
            5rtjfCQUycgJlJYWdqd+ERN9hwbXX3wZZX3PY25W5QsDwH/aJ0e5HE=
          </ds:SignatureValue>
          <ds:KeyInfo>
            <wsse:SecurityTokenReference><wsse:Reference URI="#X509certOO"
              ValueType "http://docs.oasis-open.org/wss/2004/01/oasis-200401-
              wss x509 token profile 1.0#X509"/>
            </wsse:Security okenReference>
          </ds:KeyInfo>
        </ds:Signature>
      </wsse:Security>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
      wssecurity-utility-1.0.xsd" wsu:Id="TheBody">
      <DFHDAIC4HResponse
        xmlns "http://www.DFHDAIC4H.DFHWSCFN.Response.com">
        <ca_request_id>CIC4</ca_request_id>
        <ca_return_code>0</ca_return_code>
        <ca_response_message>ORDER SUCCESSFULLY PLACED
        </ca_response_message>
        <ca_order_request>
          <ca_userid>luisl106</ca_userid>
          <ca_charge_dept>tso    </ca_charge_dept>
          <ca_item_ref_number>10</ca_item_ref_number>
          <ca_quantity_req>1</ca_quantity_req>
          <filler1>...</filler1>
        </ca_order_request>
      </DFHDAIC4HResponse>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

The <wsse:BinarySecurityToken> contains the base64binary encoding of the CICS certificate. This includes the public key that WebSphere uses to verify the signature. The rest of the SOAP headers in the response message are similar to those in the request message.

SOAP fault messages

In this section we show some of the SOAP faults that you may see when testing signature processing.

- Figure 9-35 shows the browser response when CICS is expecting a signed message but the service requester does not sign the request.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000383

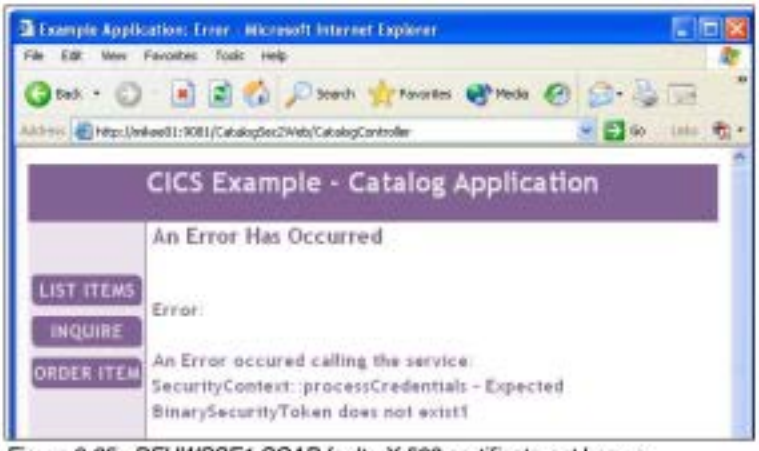


Figure 9-35 DFHWSSE1 SOAP fault - X.509 certificate not known

The SOAP fault for this message, shown in Example 9-22, highlights that CICS is expecting a BinarySecurityToken, which is not sent by WebSphere.

Example 9-22 DFHWSSE1 SOAP fault - X.509 certificate not known

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
    <faultcode>wsse:InvalidSecurity</faultcode>
    <faultstring>ERROR: Caught "XSECException" during operation:
      processMessage()</faultstring>
    <detail>
      <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
        <message>SecurityContext::processCredentials - Expected
          BinarySecurityToken does not exist</message>
        <errorcode>1</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 384/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000384

- Figure 9-36 shows the browser response when the client certificate is not associated to a user ID.

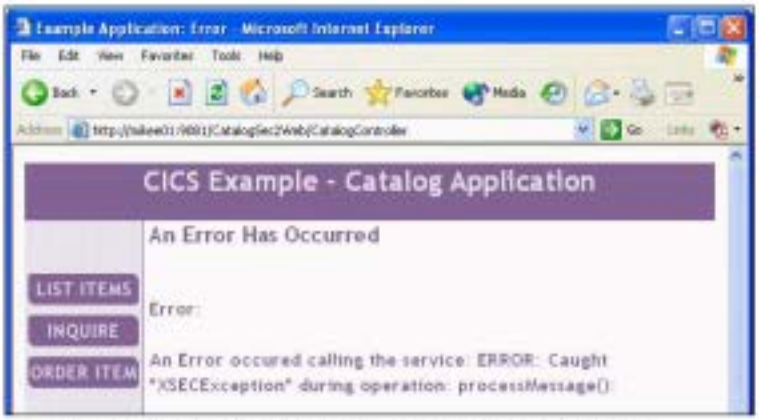


Figure 9-36 DFHWSSE1 SOAP fault - X.509 certificate not known

The SOAP fault for this message, shown in Example 9-23, highlights that there is no RACF user ID defined for the WebSphere certificate.

Example 9-23 DFHWSSE1 SOAP fault - X.509 certificate not known

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>wsse:FailedAuthentication</faultcode>
      <faultstring>ERROR: Caught *XSECException* during operation:
        processMessage()</faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
          <message>XSECKeyInfoResolverZos::extractUserId - Either no RACF user
            ID is defined for this certificate, or the certificate status is
            NOTRUST.</message>
        <errorcode>1</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 385/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000385

9.6 Encrypting a SOAP message

To provide confidentiality we can encrypt the contents of a SOAP message using XML encryption. If the SOAP message is encrypted, only a service consumer that knows the key for confidentiality can decrypt and read the message.

CICS supports XML encryption in both service provider and service requester modes.

In this section we show how a SOAP message can be encrypted by WebSphere Application Server and how the SOAP message can be decrypted by CICS. We also show how the response message can be encrypted by CICS and decrypted by WebSphere. This scenario is shown in Figure 9-37.

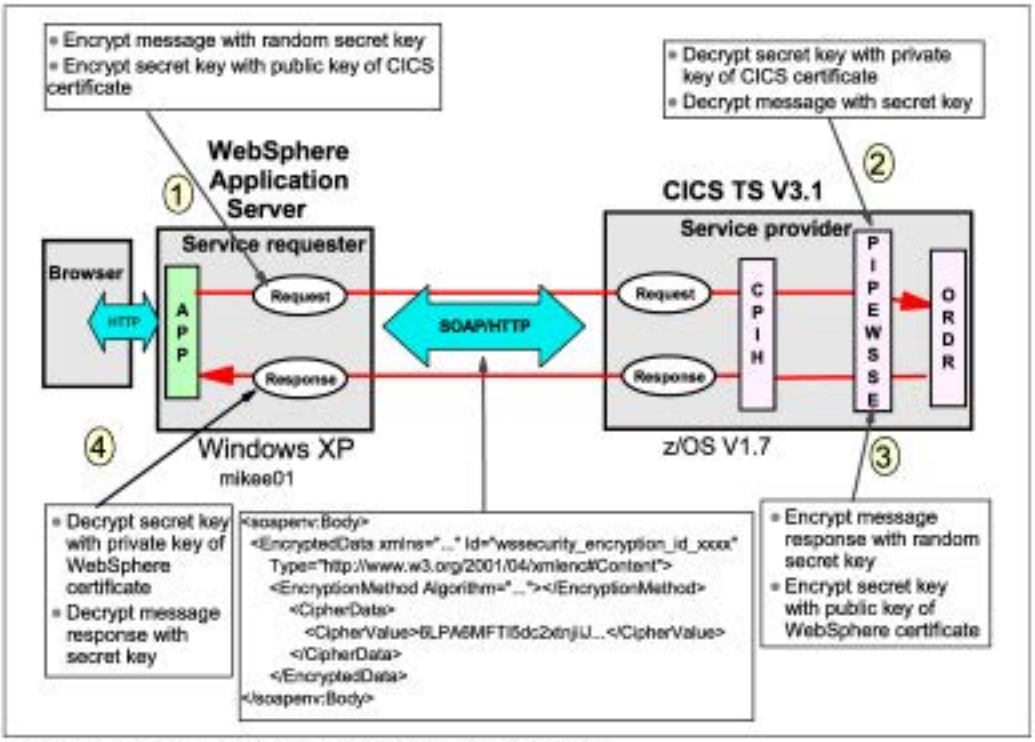


Figure 9-37 Enabling CICS WS-Security support for encryption

1. The WebSphere application server generates a random secret key and uses it to encrypt the SOAP message body. It also encrypts the secret key with the

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 386/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000386

CICS public key so only the CICS region can decrypt it. Both the encrypted data and the encrypted secret key are attached to the SOAP message.

2. CICS decrypts the secret key with its private key and uses it to decrypt the SOAP body.
3. CICS then encrypts the SOAP body of the response message with a different randomly generated secret key. It also encrypts the secret key with the public key of WebSphere so only the WebSphere application server can decrypt it.
4. The WebSphere application server decrypts the secret key with its private key and uses it to decrypt the SOAP body of the message response.

Recommendation: In our encryption scenario we use the same key pairs that we used in the signature scenario. In a production environment, however, we recommend that you use different key pairs for signature and encryption processing.

Important: There is a significant performance overhead when using WS-Security and XML encryption. See "Comparison of transport versus SOAP message security" on page 241 for recommendations on choosing between WS-Security and SSL when implementing a confidentiality security solution.

At a high level, the steps to enable support for XML encryption processing are the following:

- Configure the service requester for encryption.
- Configure CICS for encryption.
- Test the encryption test scenario.

Step-by-step details to accomplish these tasks are presented in the following sections.

Note: In our encryption test scenario we encrypt both the request and response messages. It is also possible to encrypt only the request, or to encrypt only the response. Equally, it is possible to both encrypt *and* sign a message request or response.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000387

9.6.1 Configuring the service requester for encryption

In this section we show how to configure the client J2EE application in order to send an encrypted SOAP message. At a high level the steps are:

- Import the Web service client application into AST.
- Configure the request generator for encryption
- Configure the response consumer for encryption.
- Re-deploy the Web Service client application

Import the Web service client application into AST

To configure our J2EE application for encryption, we used the IBM WebSphere Application Server Toolkit V6.1 (AST).

See "Importing the base application" on page 305 for instructions on how to import the application. For the signature test scenario, we specified

- The name of the workspace folder as:
  C:\AST\CatalogSec2\CatalogSec2Encrypt
- The name of the EAR project as:
  CatalogSec2Encrypt

Configure the request generator for encryption

Perform the following steps to configure the request generator for encryption:

1. Expand the **Dynamic Web Project** (**CatalogSec2Web**) in the Project Explorer and open (double-click) the deployment descriptor as shown in Figure 9-38

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000388

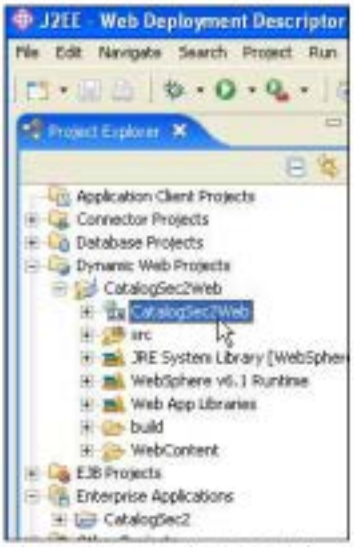


Figure 9-38 Opening the CatalogSec2Web Deployment Descriptor

2. Go to the WS Extension page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Request Generator Configuration**.
   c. Expand **Confidentiality** and click **Add**. In the Confidentiality dialog box (Figure 9-39) enter the following data:
      - Enter a Confidentiality name, for example, `conf_body`.
      - Select the order in which the encryption is generated. Multiple confidentiality parts can be specified, and you have to specify the order of generating the encryption. In our case, we select **1**.

      Note: The WS-Security runtime of WebSphere V6.1 supports multiple signature and encryption parts in one SOAP message. For multiple signature and encryption parts, you need to specify the processing order. For example, if you want to sign *and* encrypt the SOAP body, you should specify 1 in the Integrity dialog and 2 in the Confidentiality dialog.

      - Click **Add** for Message Parts, and one confidentiality part is created. The default created part is the SOAP body. We accepted the default.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 389/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000389

d. Click OK and save the configuration by pressing Ctrl+s.

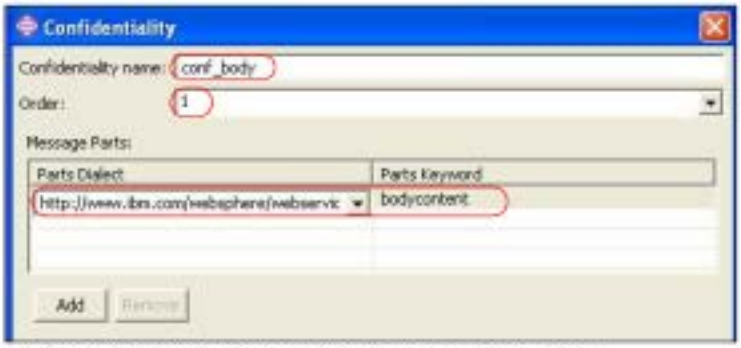


Figure 9-39 Confidentiality dialog in the WS Extensions page

3. Open the *WS Binding* page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Security Request Generator Binding Configuration**.
   c. Expand **Key Locators** and click **Add**. In the Key Locator dialog box (Figure 9-40) enter the following data:
      - Enter a Key locator name, for example, `gen_encklocator`.
      - Select **KeyStoreKeyLocator** as the Key locator class.
   d. Select **Use key store**. For key-related information refer to "Importing the base application" on page 332. Make the appropriate entries. In our case the values were:

      Password: `itso`
      Path: `C:\SG247206\keystore\was.jks`
      Type: `JKS`

   e. Click **Add** under Key and make the appropriate entries. Ours were:

      Alias: `CICSCERT`
      Key password: `itso`
      Key name: `CN=CICSCERT, O=ITSO, C=US`

      Note: This is the key name of the CICS certificate. The public key of this certificate is used by WebSphere to encrypt the request message.

   f. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000390

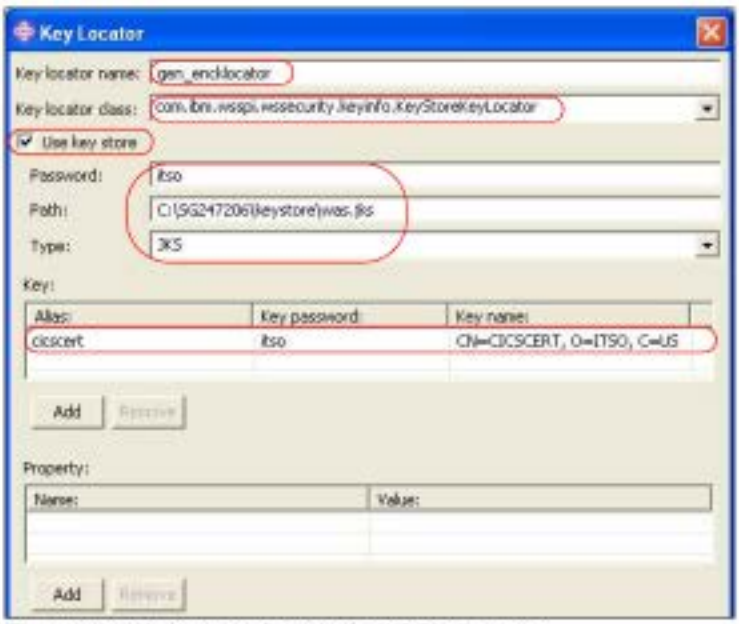


Figure 9-40 Key Locator dialog in the WS Binding page

4. Expand **Key Information** and click **Add**. In the Key Information dialog box (Figure 9-41) enter the following data:

   a. Enter a Key information name, for example, `gen_enckeyinfo`.

   b. Select **KEYNAME** as the Key information type. The Key information class is filled automatically. The Key information types are:

      - STRREF Direct reference
      - EMB Embedded reference
      - KEYID Key identifier reference
      - KEYNAME Key name reference
      - X509ISSUER x.509 issuer and serial number reference

      We specify KEYNAME because we want WebSphere to send the key information using the distinguished name of the CICS certificate.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 391/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000391

Recommendation: Use of KEYNAME for the key information type allows CICS to reference the RACF certificate directly because each certificate has a unique distinguished name and RACF provides an API for accessing the certificate using the distinguished name.

c. Select **Use key locator** and select **gen_enckIocator** from the Key locator drop-down list. Select the predefined Key name **CN=CICSCERT, O=ITSO, C=US**.

d. Click **OK**.

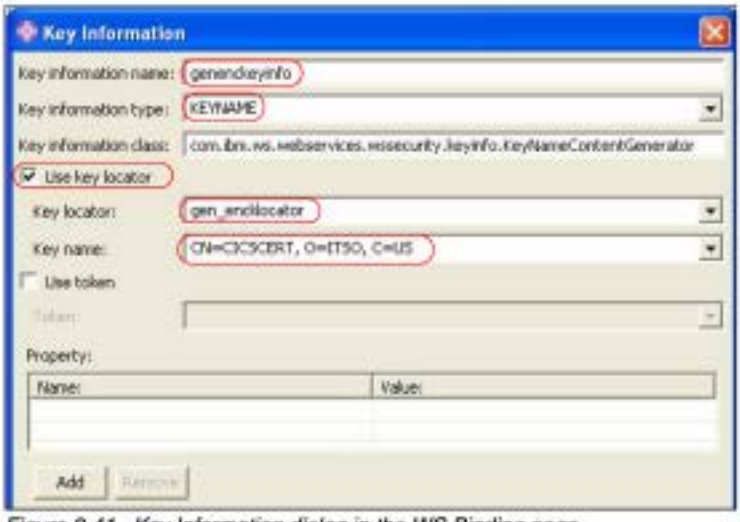


Figure 9-41 Key Information dialog in the WS Binding page

5. Expand **Encryption Information** and click **Add**. In the Encryption Information dialog box (Figure 9-42) enter the following data:

   a. Enter an Encryption Name, for example, `enc_body`.

   b. Enter a Key information name, for example, `enc_keyinfo`.

   c. Select **gen_enckey** as a Key information element.

   d. Select **conf_body** as a Confidentiality part.

   e. Allow the encryption algorithms to default.

   f. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 382/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000392

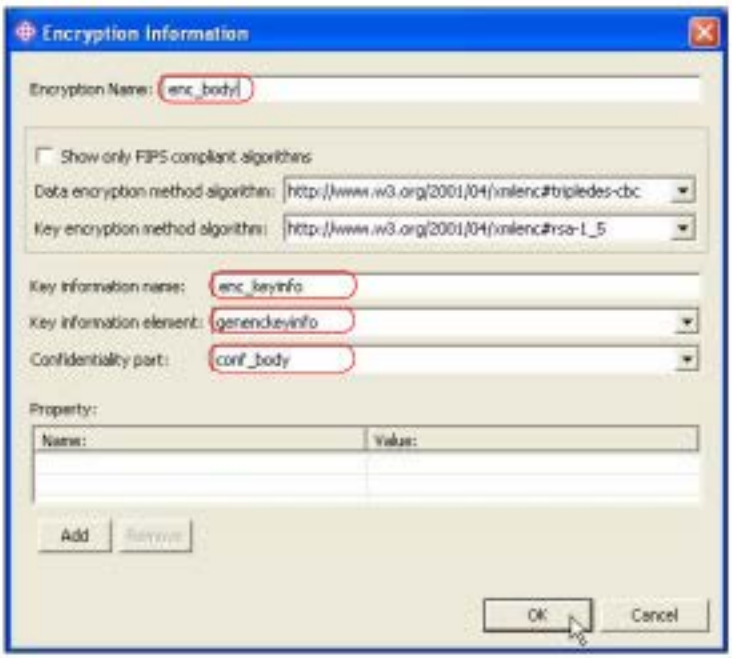


Figure 9-42 The Encryption Information dialog in the WS Binding page

6. Save the configuration by pressing Ctrl+s.

Configure the response consumer for encryption

In this section we show how to configure the client J2EE application in order to receive an encrypted SOAP message. To do this, perform the following steps:

1. Expand the **Dynamic Web Project** (**CatalogSec2Web**) in the Project Explorer and open (double-click) the deployment descriptor.
2. Go to the WS Extension page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Response Consumer Configuration**.
   c. Expand **Required Confidentiality** and click **Add**. In the Required Confidentiality dialog box (Figure 9-43) enter the following data:
      - Enter a Required confidentiality name, for example, `reqconf_body`.
      - Select the Usage type, either Required or Optional. If the usage type is Required, an unencrypted request message throws a SOAP fault. If the

Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000393

usage type is Optional, an unencrypted message is received. Select **Required**.

d. Click **Add** for Message Parts, and one message part is created. The default created part is the SOAP body. We accepted the default.

e. Click **OK** and save the configuration by pressing Ctrl+s.

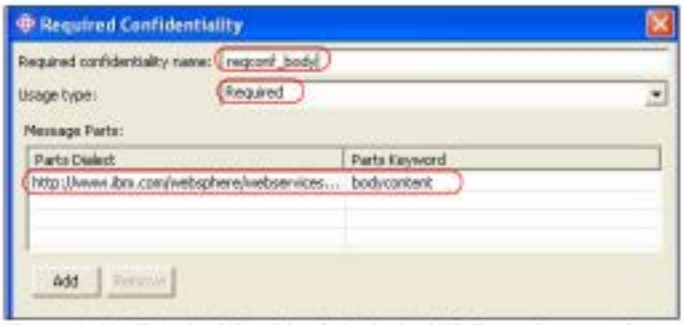


Figure 9-43 Required Confidentiality in the WS Extension page for response consumer

3. Open the WS Binding page.

   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).

   b. Expand the **Security Response Consumer Binding Configuration**.

   c. Expand **Token Consumer** and click **Add**. In the Token Consumer dialog box (Figure 9-44) enter the following data:

      - Enter a Token consumer name, for example, `con_enctcon`.
      - Select **com.ibm.wsspi.wssecurity.token.X509TokenConsumer** as the Token consumer class.
      - Select **Use value type**.
      - Select **X509 certificate token**. The Local name is selected automatically.

      Note: It is important to specify X509 certificate token for the token value type. An error occurred in our test when we specified X509 certificate token v3.

      - Select **Use jaas.config** and enter `system.wssecurity.X509BST` as the jaas.config name.
      - Select **Trust any certificate**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 394/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000394

d. Click **OK**.

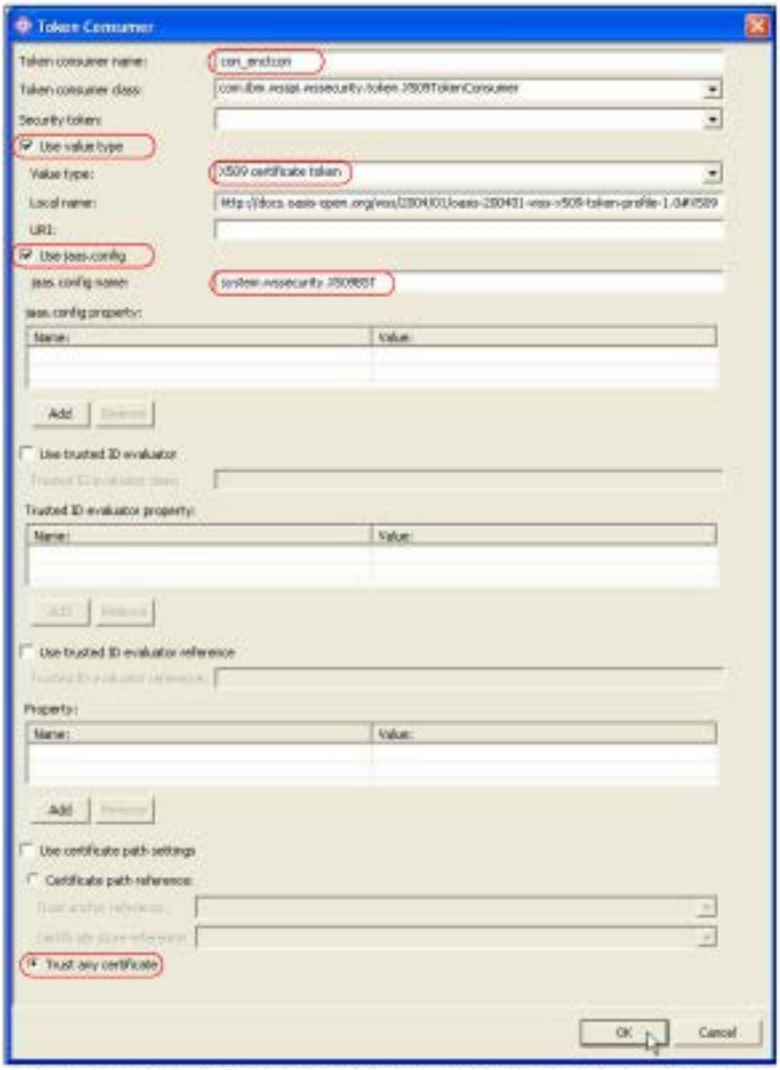


Figure 9-44 Token Consumer dialog in the WS Binding page for response consumer

4. Expand **Key Locators** and click **Add**. In the Key Locator dialog box (Figure 9-45) enter the following data:

   a. Enter a Key locator name, for example, `con_encklocator`.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000395

b. Select **KeyStoreKeyLocator** as the Key locator class.

c. Select **Use key store** and make the appropriate entries. Ours were:

Password:	`itso`
Path:	`C:\SG247206\keystore\was.jks`
Type:	`JKS`

d. Click **Add** under Key and make the appropriate entries. Ours were:

Alias:	`wascert_rsa`
Key password:	`itso`
Key name:	`CN=wascert, O=IBM, C=US`

> **Note:** This is the key name of the WebSphere certificate. The public key of this certificate is used by CICS to encrypt the response message.

e. Click **OK**.

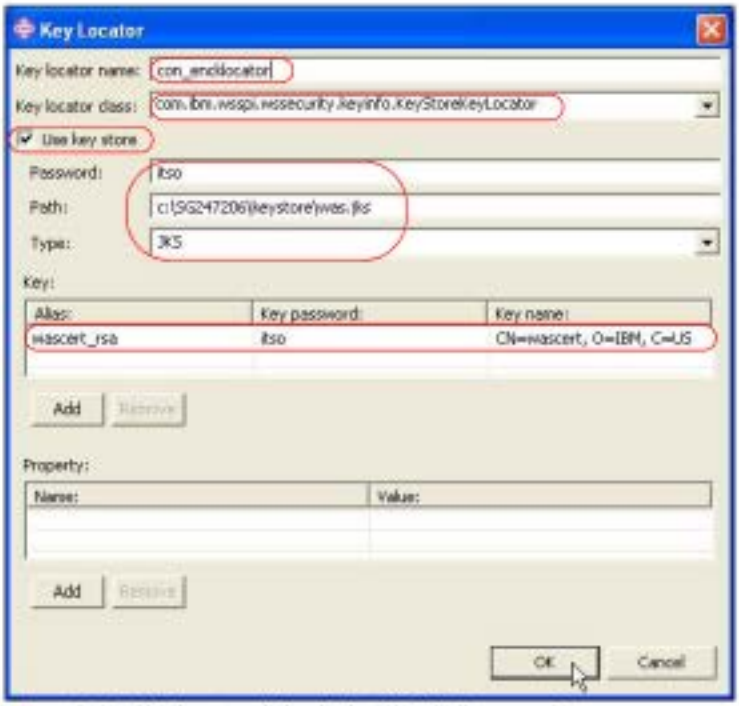


Figure 9-45 Key Locator dialog in the WS Binding page for response consumer

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
Online4victory.net
Page 396/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000396

5. Expand **Key Information** and click **Add**. In the Key Information dialog box (Figure 9-46) enter the following data:

   a. Enter a Key information name, for example, `con_enckeyinfo`.

   b. Select **STRREF** as the Key information type. The Key information class is filled automatically.

      We specify STRREF because CICS always sends the complete certificate when encrypting an outbound message. Therefore, the WebSphere response consumer must be configured to expect the WebSphere certificate in the security header of the response message.

   c. Select **Use key locator** and select **con_enclocator** from the Key locator drop-down list. Select the predefined Key name **CN=wascert, O=IBM, C=US**.

   d. Select **Use token** and select **con_enctcon** from the Token drop-down list.

   e. Click **OK**.

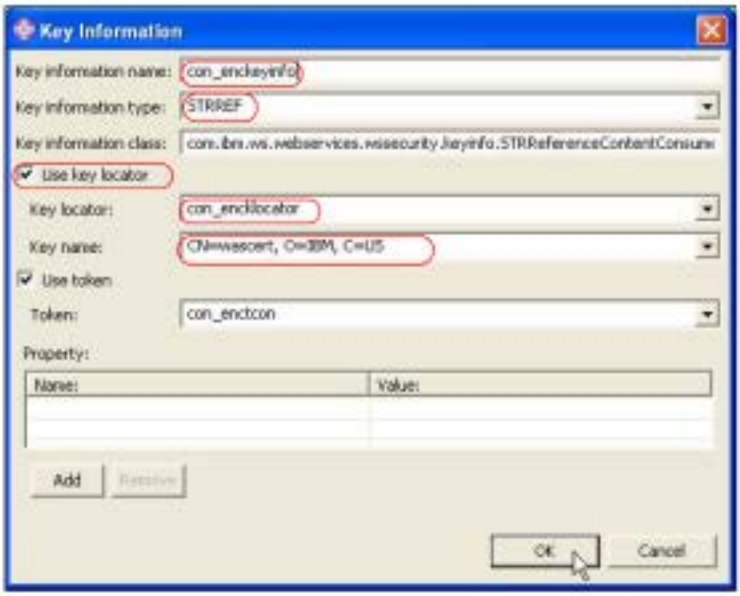


Figure 9-46 Key Information dialog in the WS Binding page for response consumer

6. Expand **Encryption Information** and click **Add**. In the Encryption Information dialog box (Figure 9-47) enter the following data:

   a. Enter a Encryption Name, for example, `enc_body`.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000397

b. Click **Add** in Encryption key information and make the appropriate entries:
   - Enter a Key information name, for example, `enc_kinfo`.
   - Select the **con_enckeyinfo** from the Key information element drop-down list.

c. Select **reqconf_body** as a Required Confidentiality part.

d. Allow the encryption algorithms to default.

e. Click **OK**.

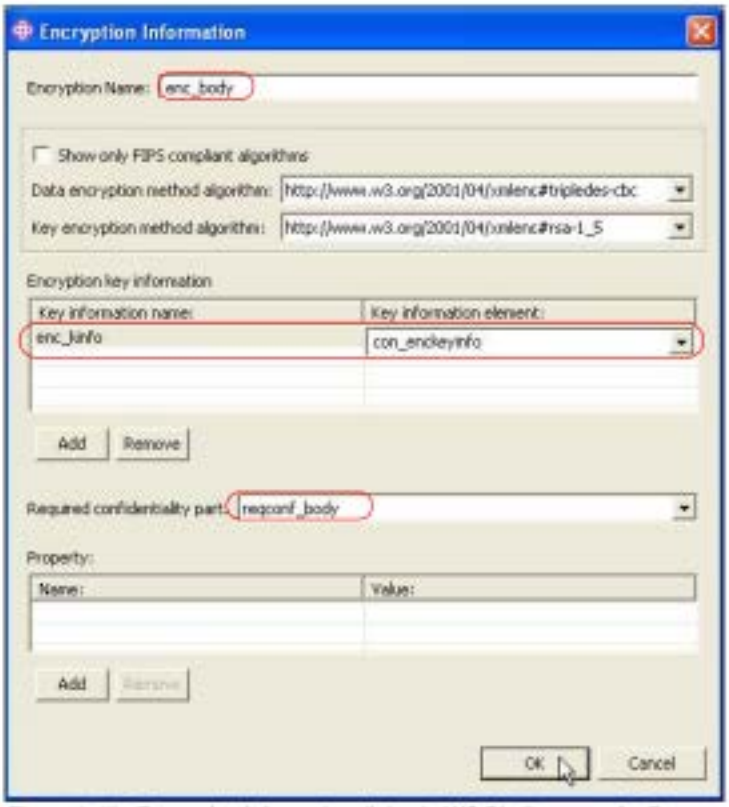


Figure 9-47 Encryption Information dialog in WS Binding page for Response Consumer

7. Save the configuration by pressing Ctrl+s.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 398/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000398

Redeploying the Web service application

After configuring an encryption security constraint for the service requester application, we exported a new EAR file called CatalogSec2_WS-Security_Encryption.ear. We followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec2_WS-Security_Encryption.ear.

For further information on configuring WS-Security in WebSphere refer to *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257.

9.6.2 Configuring CICS for encryption

In this section we show how to configure the CICS pipeline to receive and send encrypted SOAP messages.

Example 9-24 shows the pipeline configuration file that we used for the encryption test scenario. It includes the message handler DFHWSSE1, and the configuration information for the handler.

Example 9-24 Pipeline config file, ITSO_7206_wsse_signature_provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
    <transport>
      <default_transport_handler_list>
          <handler>
             <program>CHMSHSG</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
    </transport>
    <service>
      <service_handler_list>
        <handler>
          <program>SH[????]</program>
          <handler_parameter_list/>
        </handler>
        <wsse_handler>
          <dfhwsse_configuration version="1">
            <expect_encrypted_body/>
            <encrypt_body>
              <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc
              </algorithm>
              <certificate_label>WASCERT</certificate_label>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000399

```
            </encrypt_body>
          </dfhwsse_configuration>
        </wsse_handler>
      </service_handler_list>
      </terminal_handler>
        <cics_soap_1.2_handler/>
      </terminal_handler>
    </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

- The <wsse_handler> element contains a <dfhwsse_configuration> element that specifies configuration information for DFHWSSE1.
- The <expect_encrypted_body/> element indicates that the <body> of the inbound message must be encrypted. If the body of an inbound message is not correctly encrypted, CICS rejects the message with a security fault.

  We specify <expect_encrypted_body/> because we want to prevent the placeOrder service being accessed unless the body of the SOAP message is encrypted.
- The <encrypt_body/> element indicates that the <body> of the outbound message will be encrypted, and provides information about how the message is to be encrypted.
- <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm> is a child element of the <encrypt-body> element that specifies the URI of the encryption algorithm to be used for encrypting the response message. We specified the triple-DES algorithm.

  Important: The encryption algorithm must match the data encryption method algorithm specified in the WebSphere application service response consumer configuration.

- <certificate_label>WASCERT</certificate_label> is a child element of the <encrypt-body> element that specifies the label associated with the WebSphere certificate installed in RACF. This certificate contains the public key that is used by CICS to encrypt the secret (or symmetric) key. The encrypted secret key is then sent in the SOAP message response.

We completed the CICS configuration by changing the pipeline PIPEWSSE (used for the placeOrder service) to specify the pipeline configuration file that we have created for encryption processing. We used the following CEDA command:

```
CEDA ALTER PIPELINE(PIPEWSSE) GROUP(S3C1EXWS)
Configfile(/CIWS/S3C1/config/ITSO_7206_wssec_encryption_provider.xml)
```


Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000400

We then re-installed the changed pipeline resource definition

9.6.3 Testing the encryption scenario

An example of an encrypted SOAP request message for the placeOrder service is shown in Example 9-25.

Example 9-25 Encrypted SOAP request message

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
      wssecurity-secext-1.0.xsd">
      <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
        />
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <ds:KeyName>CN=CICSCERT, T=CIwss3c1-cert, OU=PSSC, O=ITSO,
            L=ENDICOTT, ST=NEW YORK, C=US</ds:KeyName>
        </ds:KeyInfo>
        <CipherData>
          <CipherValue>rh8=[y+]]]P4/g4ezibCPaxt=fxEZzQ3CSgQQ0USRMGqyEuy6b==sty9
            4CEuxDzdKV5JvSAsor8grDiExAlO2Ynn3]xc8#1Sk)1KKkV/DdiUS;C]xIyI*JK<=Da
            aT45ki/CChcruCr=9KJ2v5]oqaYAiPTHGr33C'dbcJGZjdPo=</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI="#wssecurity_encryption_id_29" />
        </ReferenceList>
      </EncryptedKey>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <EncryptedData Id="wssecurity_encryption_id_29"
      Type="http://www.w3.org/2001/04/xmlenc#Content"
      xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <CipherData>
        <CipherValue>yJ+-MZ4:kuzjfpvdskqrMHQF9Fr+y9id'9skJnK;qby;t;6eNes6=nPbtJ
          /JvpGKsEwBchUwvtL..anRujc3W6;xcX/lG+x*gAq:zoCrykzh3/lbJeHqhp44sBk-Q1
          b1V/IeaJQ9b/-.Hq:JCa/wJWWwxxZnIkd;zZqBbwQV%.tJVNJCU;:P/wcxCt/rJ'W<rp
          1RsQb4DwuZcrHOf?<//*1FRfmyL9fDKR2flDn/QENSJAcxJLZb49u7vtQm%SvDOteF11%
          iDuu4CJC19S4CfENc3xxxidt2EZYGGK0z=s.8nWrmLPQa*eHxwGuQ6PyPCDs+xsPcO4
          n2/s43Hov4VDCYgvW2%0Q6OJ4aPc5cPuJci/cH2-xyz?b2v4Vlobe!IXX8sejbuFc5yW
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000401

```
            uJoCz6uwqDpbPltUTsGUxo36YtJuu9lQ7CTcuqurs25WLTSqdZLMeRyKIygpYueT+LIIp
            ihdNuopZbC9LPag3xncb+CAnx2xwIGYLtalEE4g8rOc4EU1IY5NHqqyCXYyzy2eh+OTIC
            13TYWDU4+3/LH17rc2HOJ8NbDq5EuWptzBJ6zw*D9AcEzQY+kBvuC4Z5AHJMz/r++xOcu
            g1hNj+5xIrcI8X7o/I+MYL9IJ6saxF6Aq24zwsc?2JI8FJujWfUChIG4qcr368uI*+d
            y7h1p7Io4vpF++9CY5o7+hgh
          </CipherValue>
        </CipherData>
      </EncryptedData>
    </soapenv:Body>
</soapenv:Envelope>
```

- The header includes the mustUnderstand="1" attribute, which indicates that either this header must be processed or a SOAP fault thrown.
- The <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/> child element of the <EncryptedKey> element in the SOAP header specifies the algorithm used to encrypt the secret key.
- The <ds:KeyName> element contains the distinguished name that uniquely identifies the CICS certificate that has been used to encrypt the message. This is used by CICS to identify the CICSCERT X.509 certificate, the private key of which CICS uses to decrypt the message.
- The <CipherValue> element in the SOAP header contains the encrypted secret key that was used to encrypt the message.
- The <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/> child element of the <EncryptedData> element in the SOAP body specifies the algorithm used to encrypt the message.
- The <CipherValue> element in the SOAP body contains the encrypted message.

The corresponding encrypted response message sent by CICS is shown in Example 9-26.

Example 9-26 Encrypted SOAP response message

```
<?xml version="1.0" encoding="UTF8" standalone="no" ?>
  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis
        -200401-wss-wssecurity-secext-1.0.xsd" SOAP-ENV:mustUnderstand="1"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssp
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000402

```
curity-utility-1.0.xsd"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<wsse: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="x509certOO">MIIJ2QIJAOCAaYqOwIBAgIEN6OCzANBgkqhkiG9wOBAQUFADBN
    MQswCQYDVQQGEwJFUzELMAkGA1UECBMCQVAxCzAJBgNVBAcTAkFMMQswCQYDVQQK
    EwJFUzEPMA0GA1UECxMGQVBBTkVUMQswCQYDVQQDEwJFUzAeFw0wNjA5MjcxMTIx
    MjJaFw0wNjEwMjcxMTIxMjJaMEwxCzAJBgNVBAYTAkVTMQswCQYDVQQIEwJBUDEL
    MAkGA1UEBxMCQUwxDzANBgNVBAoTBkFQQU5FVDEPMA0GA1UECxMGQVBBTkVUMQsw
    CQYDVQQDEwJFUzCCAbcwggEsBgcqhkjOOAQBMIIBHwKBgQD9f1OBHXUSKVLfSpwu
    7OhpJ2PX0Bmi7+wH3RVDjzfZj6DbKxESZlZn2fnXWcihuMNhXJGcht2krL8vqLa1
    cJMCVKklcEVMd/JT4gUkvo7Tgo0yTPBVuPksa2ZnsV3EMYqQJennP1IKAvTrQwSR
    AjgYmLc1nOqMNFXRqi7Ox4TWLLIbFwSNDFobZ9tv1Cv1CjgcJ1in5W1DCZoq1AAo
    gYEAkOIvvGnrAi8XqGZm8FoZ1Xfo6PBVvNVIXwSSL1pvrokJ1KSkRZlMJeec3pvX
    mOuYmBQXx+6qNi5RpOyM+8FqQHnoLGT2Ranr8cjWggo1qxFoISvlBSjXvT3BZVWq
    jRoLBaB6cAgZNY2DCqdKeLLT3+5xlv1t1A6sCIDAQABoxowGDAJBgNVHRMEAjAA
    MAsGA1UdDwQEAwIE8DANBgkqhkiG9wOBAQUFAAOBgQBPTSHMA4vdoHIyexvi1H5p
    Q3n+GGdMJWAXP/DcMO+fgYaYr1l7mMBMCJDirLGrXYqjzyq2/yPAnAYFAkr+YJMO
    M7ykztCyJGchY97w75Nj/gyVBAAQjFTATMBEGA1UdDgQKBAhJYPBmS47GSTAVBgkq
    hkiG9w0BAQUFAAOBgQBPTSHMA4vdoHIPIbI8gNnUSwlsBm+xBPJiJqiXOrvYH4
    HSfpH+k4bu4fNDjO7YGSCrqSpnqGVYuHKDcPLb/c6YuDGkxr1AwCGR9+E1EJ1b
    U+MB66HqGN/qAT+glZuYrPkm2Cc0FURIQYcMMtn7+JBIc6oUp+GPQ==
</wsse:BinarySecurityToken>
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#x509cert00"
        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
        wss-x509-token-profile-1.0#X509"/>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
  <xenc:CipherData>
    <xenc:CipherValue>SLqnQ7hId/2CugP94jAGE3M5xEJDBZMBVsrF8SkW/g==Cad2
      roLHaNVcLAq==g==Cn7ihP5JifGnC6bbqxsOxJA5+SuKC4IP5zPdK4c78uEG3Eeyrc
      ZRCmYkm3MOcECD9v2xwn9J/OgfsUuXEL3+9CwvcLr49KisCC8BndvCH3=
    </xenc:CipherValue>
  </xenc:CipherData>
  <xenc:ReferenceList>
    <xenc:DataReference URI="#Enc1"/>
  </xenc:ReferenceList>
</xenc:EncryptedKey>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="Enc1" Type="http://www.w3.org/2001/04/xmlenc#Content">
    <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
    <xenc:CipherData>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000403



- The <wsse:BinarySecurityToken> contains the base64binary encoding of the WebSphere certificate. This includes the public key that WebSphere uses to decrypt the secret key.
- <wsse:SecurityTokenReference> is a child element of the <ds:KeyInfo> element which contains a reference to the BinarySecurityToken.

The rest of the SOAP headers in the response message are similar to those in the request message.

SOAP fault messages

In this section we show some of the SOAP faults that you may see when testing encryption processing.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000404

- Figure 9-48 shows the browser response when CICS is expecting an encrypted message but the service requester does not encrypt the request.

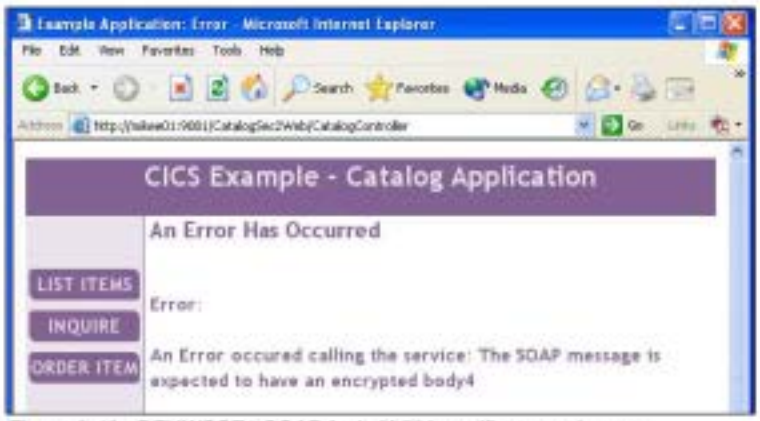


Figure 9-48 DFHWSSE1 SOAP fault: X.509 certificate not known

The SOAP fault for this message, shown in Example 9-27, highlights that CICS is expecting an encrypted message.

Example 9-27 DFHWSSE1 SOAP fault - X.509 certificate not known

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>wsse:FailedCheck</faultcode>
      <faultstring>INVALID_SOAP_REQUEST</faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
          <message>The SOAP message is expected to have an encrypted body
          </message>
          <errorcode>4</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 405/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000405

- Figure 9-49 shows the browser response when CICS is attempting to decrypt an encrypted message but ICSF is not available.

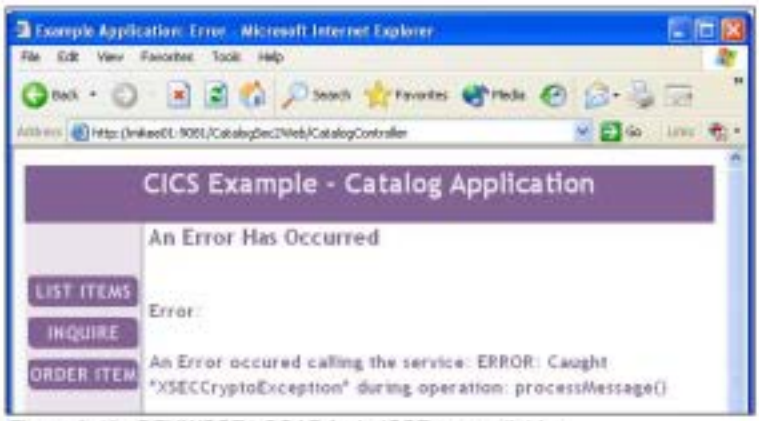


Figure 9-49 DFHWSSE1 SOAP fault: ICSF not available

The SOAP fault for this message, shown in Example 9-28, shows that there is an error creating RSA key token.

Example 9-28 DFHWSSE1 SOAP fault - ICSF not available

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>wsse:FailedCheck</faultcode>
      <faultstring>ERROR: Caught "XSECCryptoException" during operation:
        processMessage()</faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
          <message>ZosCryptoKeyRSA::privateDecrypt - Error encoding under RSA
            key</message>
          <errorcode>1</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 406/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000406

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000407

Part 4

Transaction management

In this part, we begin by providing an introduction to Web services atomic transactions. We then outline the steps for enabling WS-Atomic Transaction (WS-AT) support in CICS. Finally, we show several scenarios that demonstrate how you can synchronize WebSphere and CICS updates using the WS-AT standard.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000408

Document Id: [illegible]
Client Security: [illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000409

10

Introduction to Web services: Atomic transactions

We begin this chapter by describing an example of a classic transaction. We expect that most of our readers will be familiar with examples similar to ours. Then we map our classical transaction to a Web services *atomic transaction* as a means of introducing the new terminology that we use in this and succeeding chapters: *coordinator*, *transactional context*, *activation service*, *registration service*, and *completion protocol*. We continue by providing an overview of the flows and message contents prescribed by the three specifications upon which CICS TS V3.1 support for atomic transactions is built:

- Web Services - Addressing (WS-Addressing or WS-A)
- Web Services - Coordination (WS-Coordination or WS-C)
- Web Services - Atomic Transaction (WS-Atomic Transaction or WS-AT)

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000410

10.1 Beginner's guide to atomic transactions

We begin by describing an example of a classic transaction. For this discussion we borrow freely from the paper *Four Web Services Atomic Transaction operations: Beginner's guide to classic transactions, data recovery, and mapping to WS-Atomic Transactions* which Thomas Freund and Daniel House published on September 2, 2004, on the IBM developerWorks Web site at:

http://www-128.ibm.com/developerworks/webservices/library/ws-atmtrans1/

Not losing money is quite important. Just ask Waldo. Waldo's situation typifies the need for a transaction. Waldo uses a Web browser or an Automatic Teller Machine (ATM) to move some money from one account to another account. These accounts may be in different branches of the same financial institution, or they may be in different institutions.

It is never acceptable to Waldo for his money to disappear. Should Waldo ever doubt the safety of his money, he would probably switch financial institutions.

Waldo's money is represented by data in two databases that cooperate to ensure that the data they contain is always in a known and consistent state. That is, these two databases allow actions or tasks between them to be within a common activity or work scope as shown in Figure 10-1. Put yet another way, a single transaction can manipulate data in both databases and *something* will guarantee that only one of two possible outcomes occurs: *all* the changes are successfully made or *none* of them is made at all.

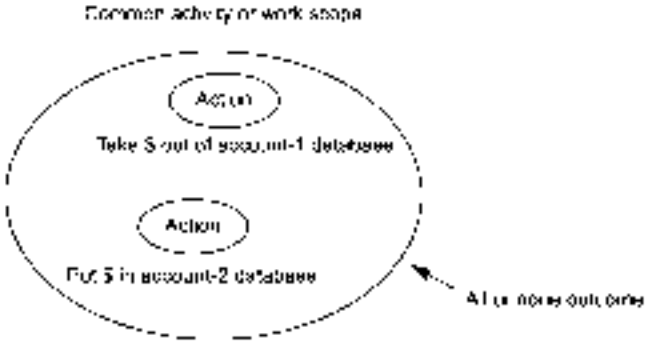


Figure 10-1 Common activity encompasses various recoverable actions

The *something* that guarantees the common outcome of all the actions is a protocol supported by both databases, and some supporting middleware. The

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000411

protocol the databases use to keep data (such as Waldo's balances) coordinated is called *two phase commit*, or simply 2PC. Our example uses a common variation of 2PC called *presumed abort*, where the default behavior in the absence of a successful outcome is to roll back or undo all the actions in the activity.

From a programming perspective, there are different ways to specify that multiple actions should be within the scope of a single transaction. One particularly clear way to specify transactional behavior is shown in Example 10-1. The code is the small piece of logic running somewhere behind the ATM Waldo is using—perhaps in the data center of one of the financial institutions involved.

Example 10-1 Pseudo-code for Waldo's transaction

```
TransferCash(fromAcct, toAcct, amount)
    BeginTransaction
    fromAcct = fromAcct - amount
    toAcct   = toAcct + amount
    CommitTransaction
Return
```

10.1.1 What is a classic transaction

For our simple purposes, a *recoverable* action is anything that modifies protected data. For example, taking money out of one of Waldo's accounts (fromAcct – fromAcct - amount) is a recoverable action that can be reversed up to the end of the transaction. A *classic* transaction, then, is just a grouping of recoverable actions, the guaranteed outcome of which is that either all the actions are taken, or none of them is taken (see Figure 10-1).

In Waldo's case, his transaction is composed of two actions, taking money out of one account and putting money into another account. It's okay for both of these actions to occur, and it's even okay if neither of these actions occurs. It's never okay for one action to occur without the other also occurring, which would result in corrupt data and either Waldo's net worth or the bank's assets disappearing or appearing from nowhere. Hence, both actions need to be within a single transaction with a single outcome: either both actions occur (a *commit* outcome), or neither action occurs (a *rollback* outcome).

Assuming no errors happen, the code in Example 10-1 shows that a commit outcome is desired. The code could just as easily have specified rollback instead of commit (for when Waldo presses the Cancel key on the ATM), which means reverse all actions in the transactional work scope (between beginning and end). The transaction monitor, which is the underlying middleware helping the code in Example 10-1 support transaction processing, would automatically specify

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000412

rollback if the program suffered an unhandled exception. Such an automatic rollback on the part of the transaction monitor is a protection mechanism to make sure that data is not corrupted. For example, even if the ATM application fails unexpectedly, the middleware will "clean up" and guarantee the outcome.

Now let's see how one common variant of 2PC, *presumed abort*, can be used to effect Waldo's transaction and move money from one account to another in a recoverable way. A key part of this illustration is to see that no matter what kind of failure occurs, data integrity is preserved and Waldo remains a loyal customer.

Figure 10-2 shows Waldo's transaction on a time line with all of the interacting components needed to execute the logic shown in Example 10-1.

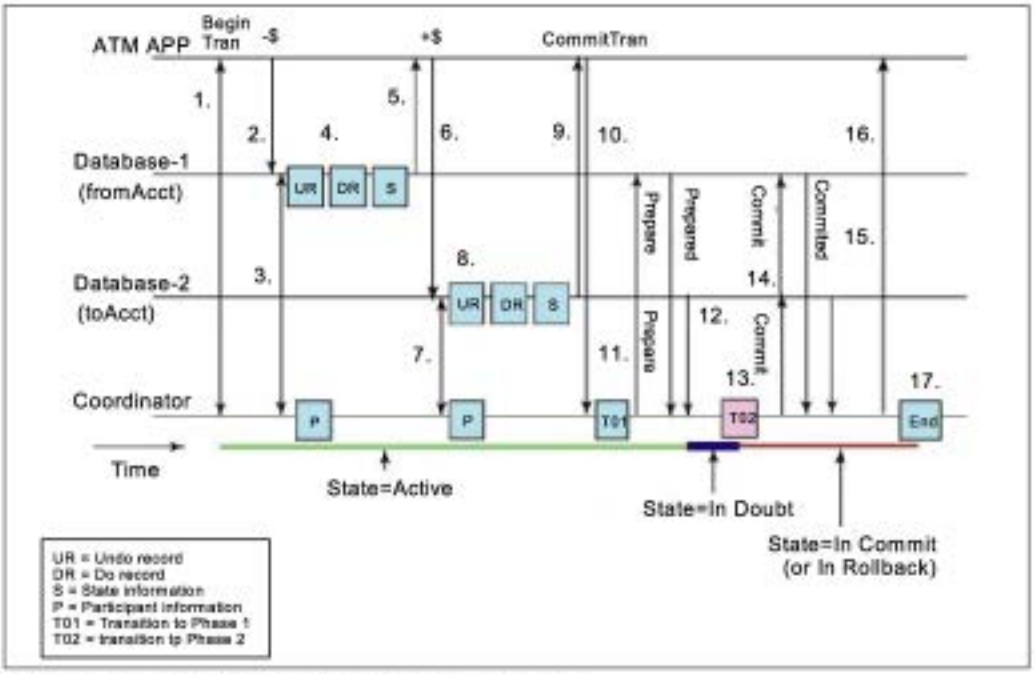


Figure 10-2 Behind the scenes of Waldo's ATM transaction

The top line represents the ATM application itself. The next two lines represent the account databases that the application manipulates. The databases will be transactional participants. The next line is a transactional coordinator, or middleware, which will orchestrate the 2PC protocol. The line at the very bottom indicates the state of Waldo's transaction at different points in time. The state of the transaction dictates recovery processing in the event of a failure.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 413/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000413

The lines for Database-1, Database-2, and Coordinator represent both time (flowing left to right) and also some key records recorded onto a recovery log. These records include images of the data before it is modified (Undo records), images of the data after it has been modified (Do records), and state information. The recovery log is used to ensure data integrity during recovery processing.

Now let's walk through Waldo's transaction. In the following discussion, when we talk about *the ATM application* take it to mean either the application itself, or some middleware supporting the application. For example, when we say the application begins a transactional scope, it could be that middleware begins the transactional scope on behalf of the application.

Here is narration to explain the numbered steps shown in Figure 10-2:

1. The ATM application indicates the beginning of a transactional scope. The Coordinator creates a context for this transaction; the context includes a unique identifier and some other information about the transaction. Importantly, this transaction context flows back to the application. The context flows with other interactions between the application and resource managers. It is the context that helps glue together a whole set of actions into one transactional activity.
2. The application takes money out of Database-1. The context (from step 1) is inserted into this flow.
3. Database-1 sees the request for action, but also sees the transactional context. Database-1 uses this context to contact the transactional Coordinator and register interest in this transaction or activity (so that the Coordinator will help Database-1 through 2PC processing later to guarantee a commit or rollback outcome of all actions). The Coordinator remembers that Database-1 is a participant in the transaction.
4. Database-1 looks at the request to modify recoverable data. It writes records to a recovery log, plus transaction state information. One record describes the database change to be made if the decision later is to commit (the Do record). The other record describes the database change to be made if the decision is to rollback (the Undo record).
   - In this case, the Undo record says *make Waldo's balance = X* and the Do record says *make Waldo's balance = X - $* (X is the amount of the balance before this transaction ever started and $ is the amount to transfer). Notice that we are only looking at the recovery log – not database files.
   - The Do records are not strictly required if Database-1 makes database file updates when the application requests it to. Instead of waiting, however, waiting to write the data can have advantages for performance and concurrency. In addition, the Do records may be used for audit or other advanced reasons. Since they are so useful, our example databases use them.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000414

5 Return to the application.

6 Similarly to step 2, the application makes a request to manipulate the other database, Database-2. The application wants to add in the amount taken out of Database-1.

7 Database-2 registers interest in the transaction with the Coordinator the same way Database-1 did. The Coordinator remembers that Database-2 is a participant in the transaction.

8 Database-2 writes Undo and Do records and state information to its recovery log, again just as Database-1 did.

9 Return to the application.

10. The application chooses to commit the transaction. The Coordinator now takes over. When **Commit** is received, the Coordinator writes a log record indicating that Phase 1 of 2PC has begun.

11. In Phase 1, the Coordinator goes down the list of all participants (Database-1 and Database-2 in this example) who expressed interest in this transaction asking each one to **Prepare**. Prepare means get ready to receive the order to either commit or rollback.

12 Database-1 and Database-2 both respond with **Prepared**, meaning that they are ready to be told the final outcome (commit or rollback all the changes made) and support it.

   - They must have committed something (at least on their logs) by this point, because responding **Prepared** means they guarantee being able to commit or rollback when told - actions up to this point were just tentative.
   - If either Database had some kind of failure preparing, it would respond **Aborted** instead of **Prepared**, and the Coordinator would broadcast **Rollback** to all participants.

13 The Coordinator forces a log record indicating a Transition to Phase 2 (TD2).

   Once this record is hardened on a log, we know and have recorded that:

   - All participants are prepared to go either way (commit or rollback).
   - The ultimate outcome of the transaction is known (commit in our example).
   - The outcome is guaranteed by recovery processing.

   If this record fails to make it to the log for any reason, the ultimate outcome will be to roll back (we are using *presumed abort* in this example). The recovery processing will enforce the outcome.

14 The Coordinator informs each participant that the decision is to commit the changes. The participants can then do whatever they need to do, such as perhaps writing the results to the real database data.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000415

15. The participants return to the Coordinator with **Committed**. Once the Coordinator knows that all the participants acknowledged the **Commit** order with **Committed**, it can forget about this transaction because the transaction was acknowledged by all to be done.

16. Return to the application.

17. At some point, since it knows the participants have succeeded in the 2PC flow by acknowledging the common outcome, the Coordinator writes an *End* indicator on its log.

10.1.2 Mapping from classic transactions to WS-Atomic Transaction

In Figure 10-2 on page 384 we did not mention how Database-1 contacted the Coordinator, nor did we specify how the application called the databases. In fact we didn't specify the mechanisms for anything to contact anything else. In the past these were mostly non-universal mechanisms that sometimes only worked between certain combinations of entities (applications, resource managers, and coordinators or transaction monitors).

The combination of Web services, WS Coordination (WS-C) and WS Atomic Transaction (WS-AT) maps all of the flows shown in Figure 10-2 on page 384 and specifies precise communications mechanisms for achieving the same results. However, instead of only working between certain combinations, the Web services based flows can work with just about anything.

Figure 10-3 illustrates how the classic flows are converted to Web services. Significantly changed steps are described following the figure. As before, when we say *application*, take it to mean the application or some helper middleware. Likewise, when we say *database*, it might mean the actual database, or some helper middleware.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000416

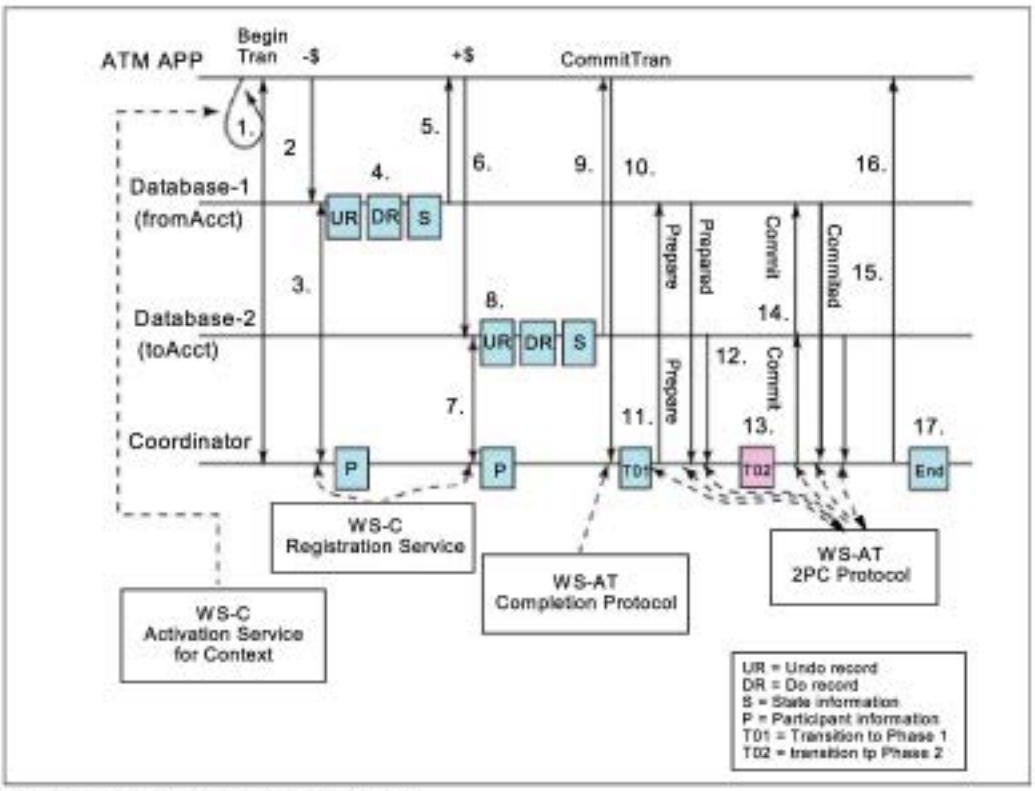


Figure 10-3 Waldo's transaction revisited

1. The application uses the Activation Service defined in WS-C to obtain a transactional context.
2. The application invokes a Web service exposed by Database-1 (alternatively, exposed by an application server that then talks to Database-1) to subtract money from Waldo's balance. The context flows along with the Web service invocation, although the application is not aware of that.
3. Database-1 uses information in the context to invoke the Registration Service defined in WS-C to register interest in this transaction.
4. No change.
5. No change.
6. The application invokes a Web service exposed by Database-2 to add money to Waldo's balance. Just like in step 2, the context flows along with the Web service invocation.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 417/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000417

7 Just like step 3, Database-2 uses information in the context to invoke the Registration Service and register interest in this transaction.

8 No change.

9 No change.

10. The application uses the Completion Protocol defined in WS-AT to indicate that it wishes to commit the transaction.

11 to 15. Databases and the Coordinator participate in 2PC flows as defined in the WS-AT 2PC Protocol.

From Figure 10-3 it is clear that atomic transactions using Web services (WS-C and WS-AT) are substantially the same as without Web services (Figure 10-2 on page 384, for example). The primary differences are almost entirely from the outside and involve *how entities communicate with each other, not the substance of what they communicate*. However, these differences in how the entities communicate have a big impact on flexibility and interoperability.

You can achieve universal interoperability with Web services because instead of changing resource manager X to interoperate with transaction monitor Y, you can change both X and Y to use Web services and then interoperate with many other resource managers and transaction monitors. So instead of two-at-a-time interoperability, or interoperability only within a specific kind of domain, n-way universal interoperability is possible.

Recovery processing using Web services between the interested parties is the same as before Web services. Resource managers are the only ones who know their resources and how to commit them or roll them back.

As an example, suppose that Database-1 fails between steps 5 and 6 in Figure 10-3. Database-1 comes back up and, just like before Web services, it reads its log, notices that it has an incomplete transaction, and realizes that it needs to contact the Coordinator. Information on how to contact the Coordinator is in the state saved on its recovery log. With Web services it will be an end point reference (as defined in WS-Addressing; see "Endpoint references" on page 391). Database-1 contacts the Coordinator at that end point reference with a message defined in WS-AT called **Replay**. Replay causes the Coordinator to resend the last protocol message to Database-1, which lets Database-1 deduce the transaction state and then apply the appropriate recovery rule. In our example the Coordinator tells Database-1 that it has no knowledge of this transaction. Database-1 therefore applies its Undo record, making the data consistent again.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000418

Important: WS-AT is a two-phase commit transaction protocol that is suitable for short duration transactions only. WS-AT is well suited for distributed transactions within a single enterprise, but is it is generally not recommended that WS-AT transactions be distributed across enterprise domains. Inter-enterprise transactions typically require a looser semantic than two-phase commit.

10.2 WS-Addressing

Figure 10-3 on page 388 shows several messages flowing:

- The application sends a message to the Activation Service asking for a transactional context.
- The Activation Service sends a response containing a transactional context to the application.
- Database-1 and Database-2 each send a **`Register`** message to the Registration Service and receives a reply.
- The application sends a **`Commit`** message to the Coordinator.
- The Coordinator sends **`Prepare`** and **`Commit`** messages to Database-1 and Database-2.
- Database-1 and Database-2 each send **`Prepared`** and **`Committed`** messages to the Coordinator.

The application, Database-1, Database-2, the Activation Service, the Registration Service, and the Coordinator are endpoints for these messages. As with messages in the everyday business world, we need a way to identify the recipient of each message, the sender of the message, what previous message (if any) the message relates to, what action we want the recipient to take, and where the recipient should send the reply (if any) to the message. Furthermore, we want to be able to do this in a way that does not depend on the transport mechanism (such as HTTP or WebSphere MQ) that we use to send the message.

To this end IBM, Microsoft, Sun™ Microsystems™, BEA, and SAP® formally submitted the *Web Services- Addressing (WS - Addressing)* specification to the World Wide Web Consortium (W3C) on 10 August 2004. You can find this specification at:

`http://www.w3.org/Submission/ws-addressing`

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 419/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000419

Note: As the specification has moved through the W3C standards process, it has been divided into three parts:

- Web Services Addressing 1.0 - Core

  This is currently a W3C Candidate Recommendation dated 17 August 2005.

- Web Services Addressing 1.0 - SOAP Binding

  This is also currently a W3C Candidate Recommendation dated 17 August 2005.

- Web Services Addressing 1.0 - WSDL Binding

  This is currently a W3C Working Draft dated 13 April 2005.

However, the WS-Coordination and WS-Atomic Transaction specifications that we discuss later in this chapter are based on the 10 August 2004 specification, and, therefore, so is CICS TS V3.1 support for Web services.

All information items defined by the 10 August 2004 specification are identified by the XML namespace URI:

```
http://schemas.xmlsoap.org/ws/2004/08/addressing
```

We associate the namespace prefix *wsa* with this namespace by using the attribute

```
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
```

The specification defines two constructs:

- Endpoint references
- Message information headers

10.2.1 Endpoint references

A Web service *endpoint* is a (referenceable) entity, processor, or resource to which Web services messages can be addressed.

An *endpoint reference* conveys the information needed to address a Web service endpoint. As we shall see later:

- An endpoint reference for the Registration Service forms part of the coordination context.
- An endpoint reference for the participant's Protocol Service forms part of the Register request.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 420/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000420

- An endpoint reference for the coordinator's Protocol Service forms part of the response to a Register request.

Example 10-2 shows the pseudo schema for an EndpointReference element.

Example 10-2 Pseudo schema for EndpointReference element

```
<wsa:EndpointReference>
   <wsa:Address>.....</wsa:Address>
   <wsa:ReferenceProperties>.....</wsa:ReferenceProperties>
   <wsa:ReferenceParameters>.....</wsa:ReferenceParameters>
   <wsa:PortType>.....</wsa:PortType>
   <wsa:ServiceName PortName="...">.....</wsa:ServiceName>
   <wsp:Policy>.....</wsp:Policy>
</wsa:EndpointReference>
```

For each child element of EndpointReference, Table 10-1 describes what the element contains, the minimum number of times the element can be used, and the maximum number of times the element can be used.

Table 10-1 Children of the EndpointReference element

Element	Description	Min	Max
Address	Contains an address URI that identifies the endpoint. This may be a network address or a logical address.	1	1
ReferenceProperties	Contains child elements each of which represents an individual reference property. The number of child elements is not limited	0	1
ReferenceParameters	Contains child elements each of which represents an individual reference parameter. The number of child elements is not limited	0	1
PortType	Contains the name of the primary portType of the endpoint being conveyed	0	1
ServiceName	Contains the name of the WSDL <service> element that contains a WSDL description of the endpoint being referenced. The service name provides a link to a full description of the service endpoint. The ServiceName element may optionally have a PortName attribute which specifies the name of the specific WSDL <port> definition in that service which corresponds to the endpoint being referenced.	0	1
Policy	Contains an XML policy element as described in WS-Policy that describes the behavior, requirements, and capabilities of the endpoint.	0	No limit

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000421

Here is the difference between a reference property and a reference parameter:

- A *reference property* is required to identify the endpoint. It is required to properly dispatch a message to an endpoint at the endpoint side of the interaction.
- A *reference parameter* is required to facilitate a particular interaction with the endpoint. It is required to properly interact with the endpoint.

Tip: In "Transaction scenarios" on page 435 we look at the messages that flow between CICS TS V3.1 and WebSphere Application Server V6.0. In these messages we see only the following children of the `EndpointReference` element:

- Address
- ReferenceProperties

Example 10-3 shows an endpoint reference for the Registration Service running in a CICS TS V3.1 region that is monitoring port 15301 on a z/OS system whose IP address is MVSG3.mop.ibm.com. The endpoint reference has two reference properties: `UOWID` and `PublicID`. We replaced the ending characters of the `PublicID` property with ... for brevity.

Example 10-3 Sample EndpointReference element for Registration Service in CICS

```
<wsa:EndpointReference
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
      xmlns:cicswsat="http://www.ibm.com/xmlns/prod/CICS/pipeline">
   <wsa:Address>
       http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
   </wsa:Address>
   <wsa:ReferenceProperties>
      <cicswsat:UOWID>BDFC52CD7C57D466</cicswsat:UOWID>
      <cicswsat:PublicId>310FD7E2E2...</cicswsat:PublicId>
   </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

Example 10-4 shows an endpoint reference for the Registration Coordinator Port running in a WebSphere Application Server V6.0 region that is monitoring port 9080 on a Windows system whose IP address is 9.100.199.156. The endpoint reference has two reference properties: `txID` and `instanceID`. We replaced the ending characters of these properties with ... for brevity.

Example 10-4 Sample EndpointReference element for Registration Service in WAS

```
<wsa:EndpointReference
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
   xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension">
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net



Page 422/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000422

```
    <wsa:Address>
       http://9.100.199.138:9080/_IBMSYSAPP/wscoor/services/Registration
       CoordinatorPort
    </wsa:Address>
    <wsa:ReferenceProperties>
       <websphere-wsat:txID>
            com.ibm.ws.wstx:0000010d:d70ad76c0000000c0000000bcbbb1a/db7e...
       </websphere-wsat:txID>
       <websphere-wsat:instanceID>
            com.ibm.ws.wstx:0000010d:d70ad26c0000000c0000000bcbbb1a/db7e...
       </websphere-wsat:instanceID>
    </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

10.2.2 Message information headers

Example 10-5 shows pseudo schema for the message information headers defined in the WS-Addressing specification.

Example 10-5 Pseudo schema for message information header elements

```
<wsa:MessageID>.....  .....</wsa:MessageID>
<wsa:RelatesTo RelationshipType="...">...</wsa:RelatesTo>
<wsa:To>.....</wsa:To>
<wsa:Action>.....</wsa:Action>
<wsa:From>.....</wsa:From>
<wsa:ReplyTo>.....</wsa:ReplyTo>
<wsa:FaultTo>.....</wsa:FaultTo>
```

Table 10-2 describes what each message information header element contains, the minimum number of times the element may be used, and the maximum number of times the element may be used.

Table 10-2 Message information header elements

Element	Description	Min	Max
MessageID	Contains a URI that uniquely identifies this message in space and time.	0 (but see Note 1)	1
RelatesTo	Contains a URI that corresponds to a related message's MessageID property. The RelatesTo element has an optional RelationshipType attribute that indicate the type of relationship this message has to the related message. The specification defines one relationship type, namely wsa:Reply When absent, the implied value of this attribute is wsa:Reply.	0	No limit

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000423

Element	Description	Min	Max
To	Contains a URI that specifies the address of the intended receiver of this message.	1	1
Action	Contains a URI that uniquely identifies the semantics implied by this message	1	1
From	Contains an endpoint reference that identifies the endpoint from which the message originated.	0	1
ReplyTo	Contains an endpoint reference that identifies the intended receiver for replies to this message.	0 (but see Note 2)	1
FaultTo	Contains an endpoint reference that identifies the intended receiver for faults related to this message.	0 (see also Note 3)	1

Table notes:

1. If ReplyTo or FaultTo is present, MessageID must be present.
2. If a reply is expected, ReplyTo *must* be present
3. When formulating a fault message, the sender *must* use the contents of the FaultTo element of the message to which the Fault reply is being sent. If the FaultTo element is absent, the sender *may* use the contents of the ReplyTo element to formulate the fault message. If both the FaultTo element and the ReplyTo element are absent, the sender *may* use the contents of the From element to formulate the fault message. The FaultTo element may be absent if the sender cannot receive fault messages.

Example 10-6 shows an example of a set of message information headers

Example 10-6 Sample message information header elements

```
<wsa:To>
   http://9.100.199.156:9080/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
</wsa:To>
<wsa:Action>
   http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register
</wsa:Action>
<wsa:MessageID>UUID-A6F0F3C1-0333477607977B5C</wsa:MessageID>
<wsa:ReplyTo>
   <wsa:Address>
      http://MYSQL.rop.ibm.com:1630/cics/wsat/RegistrationService
   </wsa:Address>
   <wsa:ReferenceProperties>
      <cicswsat:UOWID>0D7CE2C07157D466</cicswsat:UOWID>
      <cicswsat:PublicID>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000424

```
            3303D3D2D2...
        </wsaw:PublicID>
    </wsa:ReferenceProperties>
</wsa:ReplyTo>
```

The To header shows that the message is being sent to the Registration Coordinator Port running in a WebSphere Application Server V6.0 region that is monitoring port 9080 on a Windows system whose IP address is 9.100.199.156.

The Action header indicates that the sender wishes to register with the Registration Coordinator Port.

The ID of the message, `FTAI-MSG-A6P0T3C1-BB3342265277350`, uniquely identifies the message in space and time.

- `A6P0T3C1` is the VTAM APPLID of the CICS TS V3.1 region that sent the message.
- `BB3342265277350` is the abstime value returned by an EXEC CICS INQUIRE TIME issued in that CICS TS V3.1 region.

The ReplyTo header shows that the reply to this message should be sent to the Registration Service running in a CICS TS V3.1 region that is monitoring port 15301 on a z/OS system whose IP address is MVSG3.mop.ibm.com.

10.2.3 SOAP binding for endpoint references

When a SOAP message must be addressed to an endpoint, the information contained in the endpoint reference is mapped to the SOAP message by the following two rules:

- The contents of the Address element in the endpoint reference is copied to the To message information header of the SOAP message.
- Each reference property or reference parameter is added as a header block in the new message.

Example 10-7 shows how we use these rules to address a message to the CICS Registration Service endpoint whose endpoint reference is given in Example 10-3 on page 393.

Example 10-7 SOAP message addressed to CICS RegistrationService endpoint

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:cicswsa="...">
   <S:Header>
      ...
      <wsa:To>
          http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
```


Document Id: [illegible]
Confidentiality: [illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000425

```
        </wsa:To>
        <cicswsat:UOWLID>BDF0E2C0215D466</cicswsat:UOWLID>
        <cicswsat:PublicID>3109F07E2E2...</cicswsat:PublicID>
        ...
    </S:Header>
    <S:Body>
        ...
    </S:Body>
</S:Envelope>
```

10.3 WS-Coordination

In Section 10.1 "Beginner's guide to atomic transactions" on page 382 and Section 10.2. "WS-Addressing" on page 390 we see that the application, Database-1, Database-2, the Activation service, the Registration service, and the Coordinator are endpoints for messages. We also see that:

- The application sends a message to the Activation service asking for a transactional context
- The Activation service sends a response containing a transactional context to the application.
- Database-1 and Database-2 each sends a **Register** message to the Registration service and receives a reply.

We still need to define what these messages should contain.

For this purpose IBM, Microsoft, and BEA published the *Web Services-Coordination (WS - Coordination)* specification in September of 2003; they updated it in November of 2004. Arjuna Technologies Ltd., Hitachi Ltd., and IONA Technologies joined IBM, Microsoft, and BEA in publishing *Web Services-Coordination (WS-Coordination) Version 1.0* in August of 2005. You can find these at:

```
http://www-128.ibm.com/developerworks/library/specification/ws-tx
```

Although CICS TS V3.1 was developed and tested when the November, 2004 version was the current version, the nature of the differences between the November, 2004 version and the August, 2005 version is such that it may accurately be said that CICS TS V3.1 also supports the August, 2005 version.

All information items defined by the November, 2004 and August, 2005 versions are identified by the XML namespace URI:

```
http://schemas.xmlsoap.org/ws/2004/10/wscoor
```


Document Id: [illegible]
Confidential: [illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000426

We associate the namespace prefix *wscoor* with this namespace by using the attribute:

```
xmlns:wscoor="http://schemas.xmlsoap.org/ws/2004/10/wscoor"
```

The specification defines:

- A coordination service
- The following messages:
  - CreateCoordinationContext
  - CreateCoordinationContextResponse
  - Register
  - RegisterResponse

10.3.1 Coordination service

As shown in Figure 10-4, a *Coordination service* (or *Coordinator*) is an aggregation of the following services:

- Activation service

  When the application sends a `CreateCoordinationContext` element, the Activation service creates a new activity and returns its coordination context in a `CreateCoordinationContextResponse` element.

  The Coordination service may, but does not have to, support the Activation service.

  Note: Some products provide this as an external service, for others to call. CICS does not do this, and only supports the creation of coordination contexts internally, for use by the workloads that it manages.

- Registration service

  The Registration service defines a **`Register`** operation that allows a Web service to register to participate in a coordination protocol.

  The Coordination service must support the Registration service.

- A set of coordination Protocol services for each supported coordination type.

  These are defined in the specification that defines the coordination type (for example, in the WS-Atomic Transaction specification).

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000427

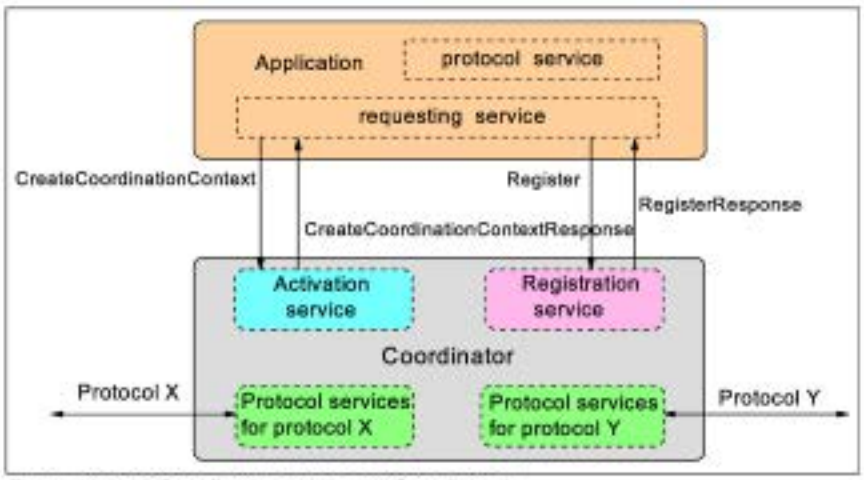


Figure 10-4 A Coordination service (or Coordinator)

10.3.2 CreateCoordinationContext

Example 10-8 shows the pseudo schema for the `CreateCoordinationContext` element.

Example 10-8 Pseudo schema for CreateCoordinationContext element

```
<wscoor:CreateCoordinationContext...>
   <wscoor:CoordinationType>.............</wscoor:CoordinationType>
   <wscoor:Expires>......................</wscoor:Expires>
   <wscoor:CurrentContext>..............</wscoor:CurrentContext>
</wscoor:CreateCoordinationContext>
```

For each child element of `CreateCoordinationContext`, Table 10-3 describes what the element contains, the minimum number of times the element can be used, and the maximum number of times the element can be used.

Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineHistory.net
Page 428/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000428

Table 10-3 Children of the CreateCoordinationContext element

Element	Description	Min	Max
CoordinationType	The unique identifier for the desired coordination type for the activity.	1	1
Expires	The expiration for the returned CoordinationContext expressed as an unsigned integer in milliseconds. Specifies the earliest point in time at which a transaction may be terminated solely due to its length of operation.	0	1
CurrentContext	The current Coordination Context.	0	1

Currently there are two specifications that define coordination types:

- WS-Atomic Transaction

  This specification defines the coordination type for atomic transactions where the results of operations are not made visible until the completion of the unit of work:

  `http://schemas.xmlsoap.org/ws/2004/10/wsat`

- WS-Business Activity

  This specification defines two coordination types for business activities:

  `http://schemas.xmlsoap.org/ws/2004/10/wsba/AtomicOutcome`

  `http://schemas.xmlsoap.org/ws/2004/10/wsba/MixedOutcome`

  Business activities have the following characteristics:

  - A business activity may consume many resources over a long duration.
  - There may be a significant number of atomic transactions involved.
  - Individual tasks within a business activity can be seen prior to the completion of the business activity, their results may have an impact outside of the computer system.
  - Responding to a request may take a very long time. Human approval, assembly, manufacturing, or delivery may have to take place before a response can be sent.
  - In the case where a business exception requires an activity to be logically undone, abort is typically not sufficient. Exception handling mechanisms may require business logic, for example in the form of a compensation task, to reverse the effects of a previously completed task.

  For example, selling a vacation is a business activity that may involve the travel agent in actions such as recording customer details, booking seats on

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000429

an aircraft, booking a hotel, booking a rental car, invoicing the customer, checking for receipt of payment, processing the payment, and arranging foreign currency.

Table 10-4 compares an atomic transaction with a business activity.

Table 10-4 Comparison of features of atomic transaction and business activity

Atomic transaction	Business activity
Short duration	Longer duration
Locks	Avoid locks
Suited for a more controlled environment	Suited for a loosely coupled environment
Classical resource manager mapping - think database (not business processes crossing business boundaries)	Business process mapping
Easier to think about and program ► Rollback or Commit ► Automatic rollback in case of abnormal termination	More complex ► Compensate
All resource managers move in one direction (everybody commits or rolls back in unison)	More flexible resource manager participation. They don't have to trust applications so much.

Note: CICS TS V3.1 does not support the WS-Business Activity specification. At this time there are no plans for future releases of CICS to support it either.

Example 10-9 shows an example of a `CreateCoordinationContext` element in which the coordination type is WS-Atomic Transaction.

Example 10-9 Sample CreateCoordinationContext element

```
<wscoor:CreateCoordinationContext>
   <wscoor:CoordinationType>
      http://schemas.xmlsoap.org/ws/2004/10/wsat
   </wscoor:CoordinationType>
   <wscoor:Expires>5000</wscoor:Expires>
</wscoor:CreateCoordinationContext>
```

10.3.3 CreateCoordinationContextResponse

The `CreateCoordinationContextResponse` element contains the `CoordinationContext` element.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 430/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000430

The CoordinationContext element contains four elements:

- A URI which identifies the CoordinationContext.
- The CoordinationType.
- The expiration period for the CoordinationContext.
- An endpoint reference for the Registration Service which is part of this Coordination service. Recall from "Endpoint references" on page 391 that this means that the CoordinationContext must contain the address of the Registration Service and may contain reference properties for the Registration Service.

Example 10-10 shows the pseudo schema for the CreateCoordinationContextResponse element.

Example 10-10 Pseudo schema for CreateCoordinationContext Response element

```
<wscoor:CreateCoordinationContextResponse>
   <wscoor:CoordinationContext>
      <wscoor:Identifier>.....................</wscoor:Identifier>
      <wscoor:CoordinationType>...............</wscoor:CoordinationType>
      <wscoor:Expires>........................</wscoor:Expires>
      <wscoor:RegistrationService>............</wscoor:RegistrationService>
   </wscoor:CoordinationContext>
</wscoor:CreateCoordinationContextResponse>
```

The application will place the CoordinationContext element within an application message to pass the coordination information to other parties. Conveying a CoordinationContext on an application message is commonly referred to as *flowing* the context. When a context is flowed as a SOAP header, the header must have the mustUnderstand attribute and the value of the mustUnderstand attribute must be true.

Note: When an application flows the context, it passes the address of its Registration Service and the coordination type.

Example 10-11 shows a CoordinationContext created by a CICS TS V3.1 region.

Example 10-11 Sample CoordinationContext created by CICS

```
<wscoor:CoordinationContext>
   <wscoor:Identifier>
      PIAT-CCON-A6P0T3C1-003322404576825C
   </wscoor:Identifier>
   <wscoor:CoordinationType
      http://schemas.xmlsoap.org/ws/2004/10/wsat
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000431

```
   </wscoor:CoordinationType>
   <wscoor:RegistrationService>
      <wsa:Address>
         http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
      </wsa:Address>
      <wsa:ReferenceProperties>
         <cicswsat:NetName>A6POT3C1</cicswsat:NetName>
         <cicswsat:Token>F0F0F0F0</cicswsat:Token>
         <cicswsat:UOWID>BCD88F2EB52B924C</cicswsat:UOWID>
      </wsa:ReferenceProperties>
   </wscoor:RegistrationService>
</wscoor:CoordinationContext>
```

Example 10-12 shows a `CoordinationContext` created by WebSphere Application Server V6.0.

Example 10-12 Sample CoordinationContext created by WebSphere

```
<wscoor:CoordinationContext>
   <wscoor:Expires>Never</wscoor:Expires>
   <wscoor:Identifier>
      com.ibm.ws.wstx:00000107d70ad2...
   </wscoor:Identifier>
   <wscoor:CoordinationType
      http://schemas.xmlsoap.org/ws/2004/10/wsat
   </wscoor:CoordinationType>
   <wscoor:RegistrationService>
      <wsa:Address>
         http://9.100.199.156:9080/_IBMSYSAPP/wscoor/services/RegistrationCoord
         inatorPort
      </wsa:Address>
      <wsa:ReferenceProperties>
         <websphere-wsat:txID xmlns:websphere-wsat="......">
            com.ibm.ws.wstx:00000107d70ad2...
         </websphere-wsat:txID>
         <websphere-wsat:instanceID xmlns:websphere-wsat="......"
            com.ibm.ws.wstx:00000107d70ad2...
         </websphere-wsat:instanceID>
      </wsa:ReferenceProperties>
   </wscoor:RegistrationService>
</wscoor:CoordinationContext>
```

Note: We noted from our tests that the content of the `Expires` element (that is, `Never`) is not an unsigned integer as required by the specification. This is planned to be changed in a future release of WebSphere Application Server.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 432/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000432

10.3.4 Register

Before we provide the details of the Register request, we consider some concepts in this section so that you don't lose sight of the 'big picture.'

The Coordinator provides the application with the Endpoint reference of its Registration service in the CreateCoordinationContextResponse. The application then knows where and how to send a Register request.

Figure 10-5 shows how Endpoint references are used during and after registration.

1. The Register message targets the Endpoint reference of the Coordinator's Registration Service and includes the Endpoint reference of the application's Protocol service as a parameter.
2. The Register Response includes the Endpoint reference of the Coordinator's Protocol service.
3. At this point, both sides have the Endpoint Reference of the other's Protocol service, so the protocol messages can target the other side.

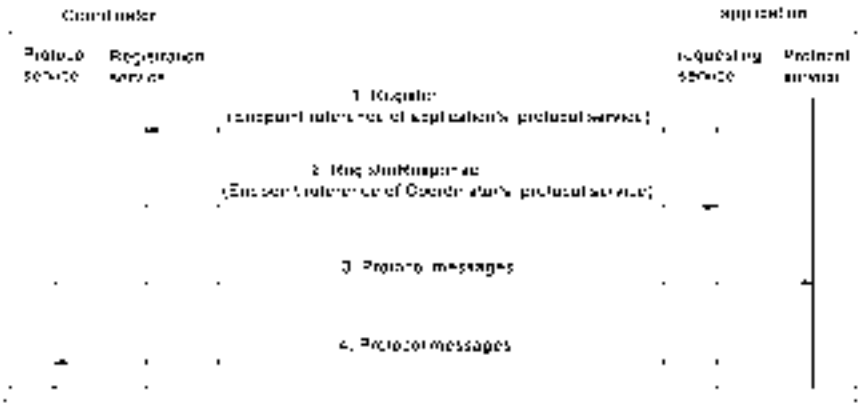


Figure 10-5 Using Endpoint references during and after registration

Now that you understand the big picture, we provide the details. Example 10-13 shows the pseudo schema for the Register element.

Example 10-15 Pseudo schema for the Register element

```
<wscoor:Register>
   <wscoor:ProtocolIdentifier>.....</wscoor:ProtocolIdentifier>
   <wscoor:ParticipantProtocolService>.....</wscoor:ParticipantProtocolService>
</wscoor:Register>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000433

The ProtocolIdentifier element contains a URI that provides the identifier of the coordination protocol selected for registration. The contents of the CoordinationType element of the CoordinationContext element determine the possible choices for the ProtocolIdentifier element as follows:

- If the CoordinationType is atomic transaction, then the ProtocolIdentifier must be one of the following:
  - http://schemas.xmlsoap.org/ws/2004/10/wsat/Completion
  - http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC
  - http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
- If the CoordinationType is business activity (either AtomicOutcome or MixedOutcome), then the ProtocolIdentifier must be one of the following:
  - http://schemas.xmlsoap.org/ws/2004/10/wsba/BusinessAgreementWithParticipantCompletion
  - http://schemas.xmlsoap.org/ws/2004/10/wsba/BusinessAgreementWithCoordinatorCompletion

The ParticipantProtocolService element contains the EndpointReference that the registering participant wants the Coordinator to use for the Protocol service.

Note: As we noted earlier, CICS TS V3.1 does not support the WS-Business Activity specification. Therefore, it does not support either the BusinessAgreementWithParticipantCompletion or the BusinessAgreementWithCoordinatorCompletion protocol identifiers.

Example 10-14 is an example of a Register element created by a CICS TS V3.1 region.

Example 10-14 Sample Register element

```
<wscoor:Register>
   <wscoor:ProtocolIdentifier>
      http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
   </wscoor:ProtocolIdentifier>
   <wscoor:ParticipantProtocolService>
      <wsa:Address>
         http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
      </wsa:Address>
      <wsa:ReferenceProperties>
         <cicswsat:UOWID>BDFC52CD7C57D466</cicswsat:UOWID>
         <cicswsat:PublicId>310FD7E2E2...</cicswsat:PublicId>
      </wsa:ReferenceProperties>
   </wscoor:ParticipantProtocolService>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 434/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000434

```
</wscoor:Register>
```

Note: CICS TS V3.1 uses a single endpoint address for its Registration service, Protocol service, and fault messages. Other products, such as WebSphere Application Server V6.0, use separate addresses for each of these.

10.3.5 Register response

Example 10-15 shows the pseudo schema for the `RegisterResponse` element.

Example 10-15 Pseudo schema for the RegisterResponse element

```
<wscoor:RegisterResponse>
   <wscoor:CoordinatorProtocolService>.....</wscoor:CoordinatorProtocolService>
</wscoor:RegisterResponse>
```

The `CoordinatorProtocolService` element contains the `EndpointReference` that the Coordination service wants the registered participant to use for the Protocol service.

We note, once again, that when the application has received the `RegisterResponse`, each side has the `EndpointReference` of the other's Protocol Service.

10.3.6 Two applications with their own coordinators

In this section we see how two application services (App1 and App2) with their own coordinators (Coordinator A and Coordinator B) interact as an activity propagates between them. Coordinator A provides Activation service ASa, Registration service RSa, and Protocol service Pa. Coordinator B provides Activation service ASb, Registration service RSb, and Protocol service Pb. Figure 10-6 shows the two applications with their own coordinators.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 435/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000435

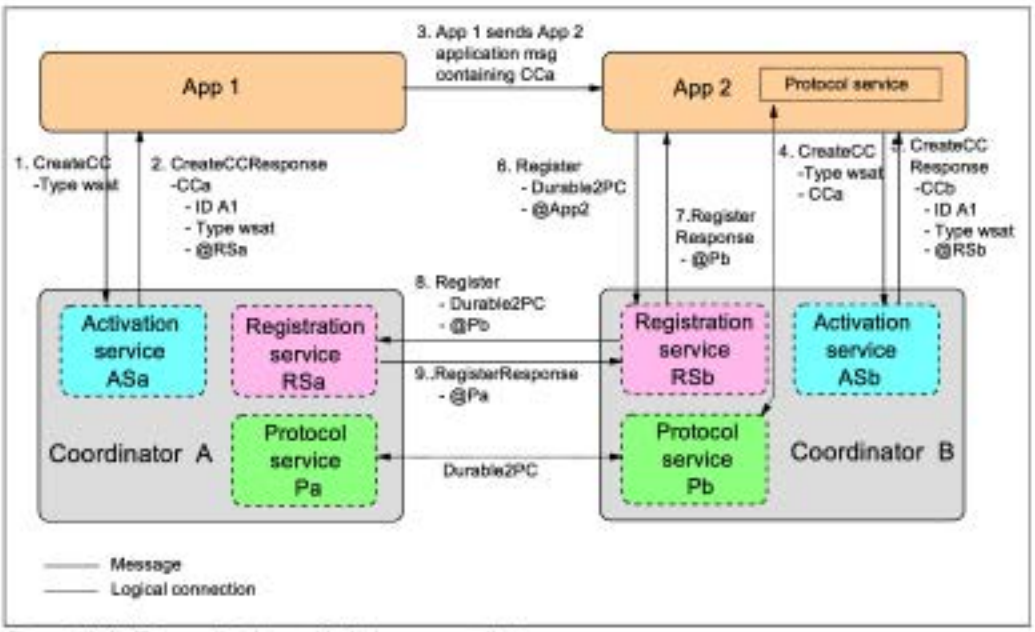


Figure 10-6 Two applications with their own coordinators

1. App1 sends to ASa a `CreateCoordinationContext` element that specifies a `CoordinationType` of wsat.
2. ASa sends to App1 a `CreateCoordinationContextResponse` element that includes `CoordinationContext` CCa. CCa contains an `Identifier` element whose content is A1, a `CoordinationType` element whose content is wsat, and a `RegistrationService` element that contains an `EndpointReference` to Coordinator A's Registration service RSa.
3. App1 then sends to App2 a SOAP message that has a SOAP header that contains CCa and that has a `mustUnderstand` attribute with a value of `True`.
4. App2 prefers Coordinator B, so it sends to ASb a `CreateCoordinationContext` element that specifies a `CurrentContext` of CCa.
5. Coordinator B creates its own `CoordinationContext` element CCb that contains the same `Identifier` and `CoordinationType` as CCa but with an `EndpointReference` to its own RegistrationService RSb.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 436/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000436

The *WS-Atomic Transaction* specification which we discuss in detail in "WS-Atomic Transaction" on page 408 states that:

If the CreateCoordinationContext request includes the CurrentContext element, then the target coordinator is interposed as a subordinate to the coordinator stipulated inside the CurrentContext element.

- If the CreateCoordinationContext request does not include a CurrentContext element, then the target coordinator creates a new transaction and acts as the root coordinator.

6. App2 determines the coordination protocols supported by the wsat coordination type and then sends a Register element to RSb. This Register element contains a ProtocolIdentifier of Durable2PC and an EndpointReference to App2's Protocol service.
7. RSb sends back to App2 a RegisterResponse element that contains an EndpointReference to Protocol service Pb. This forms a logical connection (which the Durable2PC protocol can use) between the Endpoint reference for App2's Protocol service and the Endpoint reference for Coordinator B's Protocol service Pb.
8. This registration causes Coordinator B to forward the registration on to Coordinator A's Registration service RSa.
9. RSa sends back to Coordinator B a RegisterResponse element that contains an EndpointReference to Protocol Service Pa. This forms a logical connection between the EndpointReferences for Pa and Pb that the Durable2PC protocol can use.

10.3.7 Addressing requirements for WS-Coordination message types

CreateCoordinationContext and Register messages:

- *Must* include a wsa:MessageID header
- *Must* include a wsa:ReplyTo header

CreateCoordinationContextResponse and RegisterResponse messages:

- *Must* include a wsa:RelatesTo header that specifies the MessageID from the corresponding request message

10.4 WS-Atomic Transaction

As we discussed in Section 10.3, "WS-Coordination" on page 397, the WS-Coordination specification defines an extensible framework for defining coordination types. The WS-Atomic Transaction specification builds on

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000437

WS-Coordination by providing the definition of the atomic transaction coordination type.

Atomic transactions have an *all-or-nothing* property. The actions taken prior to commit are only tentative (that is, not persistent and not visible to other activities). When an application finishes, it requests the Coordinator to determine the outcome for the transaction. The Coordinator determines if there were any processing failures by asking the participants to vote. If the participants all vote that they were able to execute successfully, the Coordinator commits all actions taken. If a participant votes that it needs to abort or a participant does not respond at all, the Coordinator aborts all actions taken. Commit makes the tentative actions visible to other transactions. Abort makes the tentative actions appear as if the actions never happened.

IBM, Microsoft, and BEA published the *Web Services- Atomic Transaction (WS - Atomic Transaction)* specification in September, 2003; they updated it in November, 2004. Arjuna Technologies Ltd., Hitachi Ltd., and IONA Technologies joined IBM, Microsoft, and BEA in publishing *Web Services - Atomic Transaction (WS - Atomic Transaction) Version 1.0* in August, 2005. You can find these at:

```
http://www-128.ibm.com/developerworks/library/specification/ws-tx
```

Although CICS TS V3.1 was developed and tested when the November, 2004, version was the current version, the nature of the differences between the November, 2004 version and the August, 2005 version is such that it may accurately be said that CICS TS V3.1 also supports the August, 2005 version.

All information items defined by the November, 2004 and August, 2005 versions are identified by the XML namespace URI:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat
```

We associate the namespace prefix `wsat` with this namespace by using the attribute:

```
xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
```

The WS-AT specification defines the following protocols for atomic transactions:

- Completion
- Volatile Two-Phase Commit
- Durable Two-Phase Commit

Document ID: [illegible]
Confidentiality: [illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000438

10.4.1 Completion protocol

The `Completion` protocol is used by an application to tell the Coordinator to try to either commit or abort an atomic transaction. The `Completion` protocol initiates commitment processing. Based on each protocol's registered participants, the Coordinator begins with Volatile 2PC and then proceeds through Durable 2PC. After the transaction has completed, a status (Committed or Aborted) is returned to the application.

An initiator registers for this protocol by specifying the following URI for the contents of the `ProtocolIdentifier` element in the `Register` element:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat/Completion
```

Figure 10-7 illustrates the protocol abstractly.

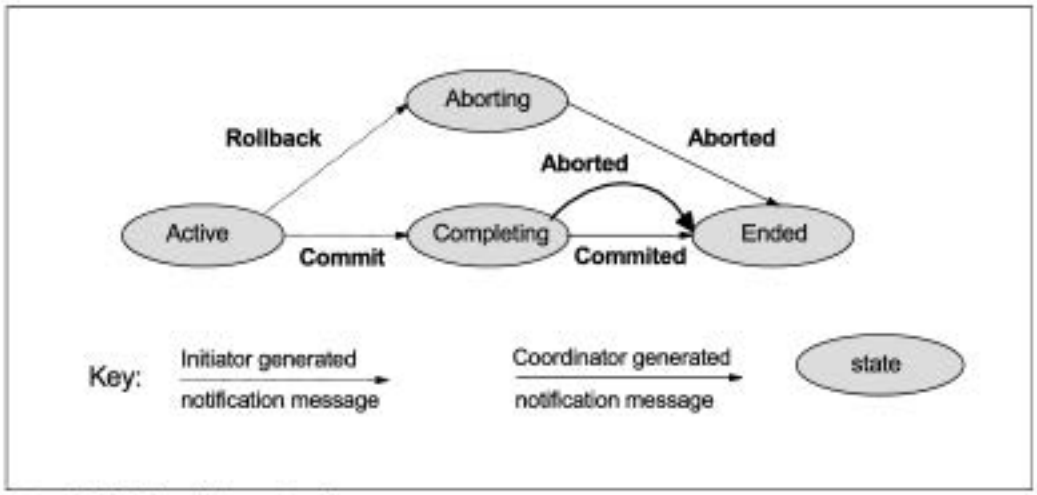


Figure 10-7 Completion protocol

The initiator generates:

- Commit

  Upon receipt of this notification, the Coordinator knows that the initiator has *completed* application processing and that it should attempt to commit the transaction.

- Rollback

  Upon receipt of this notification, the Coordinator knows that the initiator has *terminated* application processing and that it should abort the transaction.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 439/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000439

The Coordinator generates:

- Committed

  Upon receipt of this notification, the initiator knows that the Coordinator reached a decision to commit.

- Aborted

  Upon receipt of this notification, the initiator knows that the Coordinator reached a decision to abort.

10.4.2 Two-Phase Commit protocol

The Two-Phase Commit (2PC) protocol defines how multiple registered participants reach agreement on the outcome of an atomic transaction. The 2PC protocol has two variants: Volatile 2PC and Durable 2PC.

Participants managing volatile resources such as a cache should register for this protocol by using the following protocol identifier:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC
```

Participants managing durable resources such as a database should register for this protocol by using the following protocol identifier:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
```

Note: When CICS TS V3.1 is a participant in an atomic transaction, it always requests the Durable2PC protocol when it sends a Register request. When CICS TS V3.1 is the coordinator of an atomic transaction, it will tolerate a Register request for Volatile2PC but it will treat it as a Durable2PC request.

After receiving a **`Commit`** notification in the Completion protocol, the root Coordinator begins the Prepare phase of all participants registered for the Volatile 2PC protocol. All participants registered for this protocol must respond before a Prepare is issued to a participant registered for the Durable 2PC protocol. We illustrate this in Figure 10-8, where participants P1 and P3 registered for the Volatile 2PC protocol and participant P2 registered for the Durable 2PC protocol. Both P1 and P3 must respond to the **`Prepare`** notification before the Coordinator can send **`Prepare`** to P2.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 44of6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000440

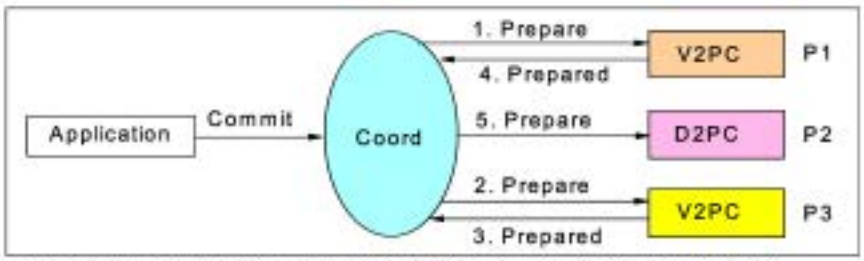


Figure 10-8 Mixture of participants registered for Durable 2PC and Volatile 2PC

Upon successfully completing the prepare phase for Volatile 2PC participants, the root Coordinator begins the Prepare phase for Durable 2PC participants. All participants registered for this protocol must respond `Prepared` or `ReadOnly` before a Commit notification is issued to a participant registered for either protocol. A volatile participant is not guaranteed to receive a notification of the transaction's outcome.

Figure 10-9 illustrates the 2PC protocol abstractly.

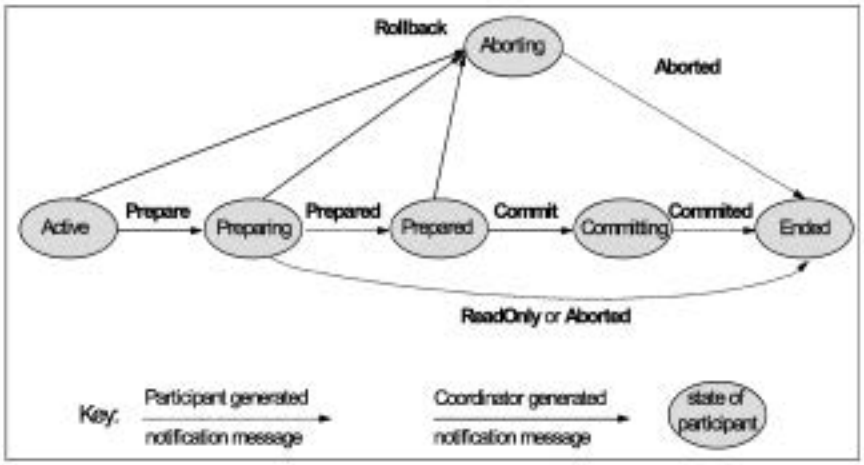


Figure 10-9 Two-Phase Commit protocol

The Coordinator generates:

- **`Prepare`**

  Upon receipt of this notification, the participant should enter phase 1 and vote on the outcome of the transaction.

  - If the participant has already voted, it should resend the same vote.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 441/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000441

– If the participant does not know of the transaction, it must vote to abort.

- **Rollback**

  Upon receipt of this notification, the participant should abort and forget the transaction. This notification can be sent in either phase 1 or phase 2. Once sent, the Coordinator may forget all knowledge of this transaction.

- **Commit**

  Upon receipt of this notification, the participant should commit the transaction. This notification can only be sent after phase 1, and if the participant voted to commit. If the participant does not know of the transaction, it must send a **Committed** notification to the Coordinator.

The participant generates:

- **Prepared**

  The participant is prepared and votes to commit the transaction.

- **ReadOnly**

  The participant votes to commit the transaction and has forgotten the transaction. The participant does not wish to participate in phase two.

  Suppose, for example, that the participant received an account number that it could not match to an entry in a database. It might return an error to the requesting application, but, having registered as a participant in the atomic transaction, it would then go on to be coordinated during 2PC processing. When the Coordinator sends **Prepare**, the participant replies **ReadOnly** and then terminates without waiting for the **Commit**. The Coordinator, on receipt of the **ReadOnly**, would then delete its own record of the interaction with the participant and would not attempt to send a **Commit** to it.

- **Aborted**

  The participant has aborted, and forgotten, the transaction.

- **Committed**

  The participant has committed the transaction. The Coordinator may safely forget that participant.

- **Replay**

  The participant has suffered a recoverable failure. The Coordinator should resend the last appropriate protocol notification.

10.4.3 Two applications with their own coordinators (continued)

In Section 10.3.6, "Two applications with their own coordinators" on page 406 we saw how two application services (App1 and App2) with their own coordinators

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000442

(Coordinator A and Coordinator B) used the WS-Coordination specification to interact as an activity propagated between them. Figure 10-10 shows how they use the WS-AT specification to complete their global unit of work.

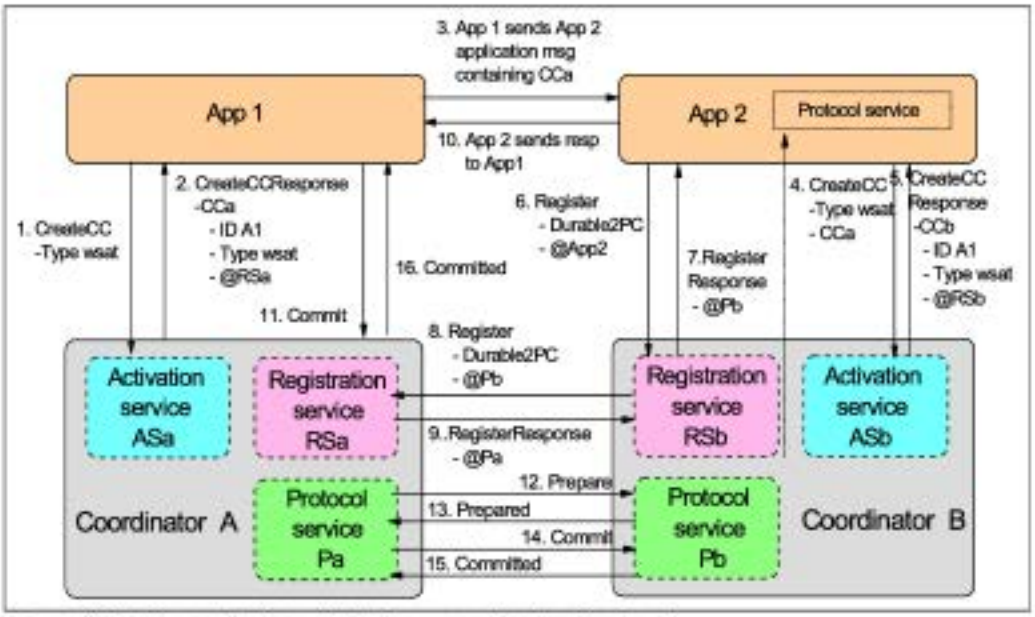


Figure 10-10 Two applications with their own coordinators (continued)

After completing its work, App2 sends its response back to App1 (step 10 in Figure 10-10). When App1 completes its work, its sends a **Commit** message to Coordinator A (step 11). This causes Coordinator A's Durable 2PC Protocol service Pa to send a **Prepare** notification to Coordinator B's Durable 2PC Protocol service Pb (step 12). We assume that Pb responds with a **Prepared** notification. If Pa encounters no other errors, it makes the decision to commit the atomic transaction and sends a **Commit** notification to Pb (step 14). Pb returns a **Committed** notification and then completes its updates before terminating. When Coordinator A receives this notification, it commits its own updates and notifies App1 of the outcome (step 16).

10.4.4 Addressing requirements for WS-AT message types

The messages defined in the WS-AT specification are *notification* messages; that is, they are *one-way* messages. There are two types of notification messages:

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 443/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000443

- A notification message is a *terminal* message when it indicates the end of a coordinator/participant relationship. **Committed**, **Aborted**, and **ReadOnly** are terminal messages.
- A notification message is a *non-terminal* message when it does not indicate the end of a coordinator/participant relationship. **Commit**, **Rollback**, **Prepare**, **Prepared**, and **Replay** are non-terminal messages.

Non-terminal notification messages *must* include a `wsa:ReplyTo` header.

10.4.5 CICS TS V3.1 and resynchronization processing

We have completed our discussion of the WS-AT specification. Unfortunately, the current version of the specification does not completely cover all of the issues surrounding the use of the 2PC protocol. In particular, it does not completely describe the resynchronization processing that should take place following a failure in one of the systems involved in the 2PC protocol or in the network connections that link the systems together. The only thing that the specification mentions relating to resynchronization is the **Replay** notification. Therefore, in this section we describe some aspects of how CICS TS V3.1 handles resynchronization processing for transactions that use the 2PC protocol.

Note: For the sake of brevity we do not describe all of the possible issues. For example, we do not describe:

- What happens when resync processing is driven from both sides and a *race* condition results
- What happens when a resync request fails

Network failures can result in messages not being delivered in a timely manner. System failures prevent processing altogether until a restart takes place.

Within the 2PC processing sequence there is a period of time, known as the *in-doubt* window, during which one system is unable to complete processing because it does not know what the other system has done. The distributed UOW is said to be in-doubt when:

- A participant Protocol service has replied **Prepared** in response to a **Prepare** notification, *and*
- Has written a log record of its response to signify that it has entered the in-doubt state, *and*
- Does not yet know the decision of its coordinator (to **Commit** or to **Rollback**).

Barring system or network failures, the UOW remains in-doubt until the coordinator issues either the **Commit** or **Rollback** request as a result of responses

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000444

received from all UOW participants. If a failure occurs that causes loss of connectivity between a participant and its coordinator, the UOW remains in-doubt until either:

- Recovery from the failure has taken place and synchronization can resume *or*
- The in-doubt waiting period is terminated by some built-in control mechanism, and an arbitrary (heuristic) decision is then taken (to commit or back out).

Note that while the UOW remains in-doubt, the recoverable resources that it owns remain locked.

If a system or network failure occurs during the in-doubt window, additional steps must be taken to ensure that the updates are completed in a consistent manner by both systems. This is known as *resynchronization processing*.

Previous releases of CICS provided a Recovery Manager that dealt with resynchronization processing for distributed workloads that made use of *VTAM®* networks or that used *MRO* connections. These releases dealt with failures during the in-doubt window in one of three ways:

- Automatic heuristic decision

  You could cause CICS to make an automatic heuristic decision by specifying the WAIT, WAITTIME, and ACTION attributes on a TRANSACTION definition as follows:

  - If you set the WAIT attribute to NO, then CICS took whatever action was specified on the ACTION attribute (either COMMIT or BACKOUT) *immediately*.
  - If you set the WAIT attribute to YES and the WAITTIME attribute to a non-zero value, then CICS took whatever action was specified on the ACTION attribute after waiting for the amount of time specified in WAITTIME (assuming normal recovery and resynchronization had not already taken place).

- Manual heuristic decision

  You could force an in-doubt UOW to complete by issuing a CEMT SET UOW(*uowid*) |COMMIT | BACKOUT| command or its EXEC CICS equivalent.

- Automatic resynchronization

  If you set the WAIT attribute to YES and the WAITTIME attribute to 00.00.00, the transaction would wait until it could communicate with its partner system, after which it could either explicitly request that the message it was waiting for be sent again, or it could resend the last message that it generated.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000445

CICS TS V3.1 extends the Recovery Manager for use by WS-AT workloads. CICS applications that form part of a WS-AT workload can be controlled by any of these mechanisms. However, automatic resynchronization is somewhat different for WS-AT workloads.

The principle difference arises from the fact that WS-AT processing takes place over a TCP/IP network.

- Other forms of distributed transactions make use of communication mechanisms such as VTAM, and resynchronization across a VTAM network can be triggered when the connection between a pair of systems is re-established.
- CICS does not currently support TCP/IP connections in the same way that it does its VTAM connections, and so CICS can only drive resynchronization of WS-AT requests when a region starts.

Important: CICS only drives resynchronization of WS-AT requests during a CICS region restart.

During any type of startup except an initial start, CICS reads the system log to discover any units of work that were in-doubt when the region previously shut down or failed. While reading through the log, CICS may find that it has outstanding units of work that indicate they were involved in an atomic transaction. These log records also indicate whether the UOW was acting as a coordinator or a participant.

- Coordinator

  If it is a coordinator and the log record indicates that the UOW was waiting for a **`Committed`** or **`Aborted`** response from a participant when CICS shut down, then the UOW is reactivated (unshunted) and sends out its decision message (**`Commit`** or **`Rollback`**) to the participant identified in the log record.

  - If a response is received, then the UOW completes its processing and terminates.
  - If a response is not received before the coordination UOW times out, then CICS shunts the UOW (moves it aside for processing later on). The UOW then persists until another resynchronization attempt takes place or until someone manually forces it to complete. (The coordination UOW times out after 30 seconds, a value set internally by CICS).

- Participant

  If it is a participant and the log record indicates that the UOW had voted in response to a **`Prepare`** message and was waiting for a **`Commit`** or **`Rollback`** decision from its coordinator when communication was lost, then the UOW is

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 446/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000446

reactivated, sends a **Replay** message to its coordinator, and once again waits for the decision message to arrive.

If the decision message is then received, the participant acts on it and sends a **Committed** or **Aborted** message back to the coordinator before terminating.

- If the decision message does not arrive before the participant UOW times out, then CICS shunts the participant UOW. The UOW then persists until another resynchronization attempt takes place or until someone manually forces it to complete.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000447

11

Enabling atomic transactions

We begin this chapter by showing you how to enable atomic transactions in CICS. We start with the simple case where we have a service requester application running in CICS AOR1 invoking a service provider application running in CICS AOR2. Then we move to the more elaborate case where two CICSplexes are working together, one acting as service requester and the other as service provider. We conclude the chapter by showing you how to enable atomic transactions in WebSphere Application Server.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000448

11.1 Enabling atomic transactions in CICS

We recognize that it is not likely that many customers will choose to use WS-AT for workloads distributed entirely within CICS. More typically, CICS would participate in an atomic transaction as a Web service provider with the service requester running under the control of another product such as WebSphere Application Server V6 or later. Customers might also use a CICS transaction acting as a Web service requester to participate in an atomic transaction with a service provider running under the control of another product.

Nevertheless, we start with the special case of one CICS region acting as a service requester participating in an atomic transaction with another CICS region acting as a service provider. This will illustrate the capacity of CICS to undertake both the role of a coordinator of an atomic transaction and the role of a participant in an atomic transaction.

11.1.1 CICS to CICS configuration

Figure 11-1 shows two CICS regions: AOR1 and AOR2. A service requester application running in AOR1 invokes a service provider application running in AOR2.

In AOR1 the request passes through a pipeline that contains a CICS-provided message handler module (either DFHPISN1, if SOAP 1.1 is being used; or DFHPISN2, if SOAP 1.2 is being used). DFHPISNx invokes the CICS-provided header processing program DFHWSATH. DFHWSATH adds a SOAP header containing a CoordinationContext to each message that it sends out.

In AOR2 the request passes through a pipeline that supports the Web service that AOR1's application is calling and also invokes DFHPISNx. DFHPISNx invokes the header processing program DFHWSATH when it detects a SOAP header that contains a `CoordinationContext`.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000449

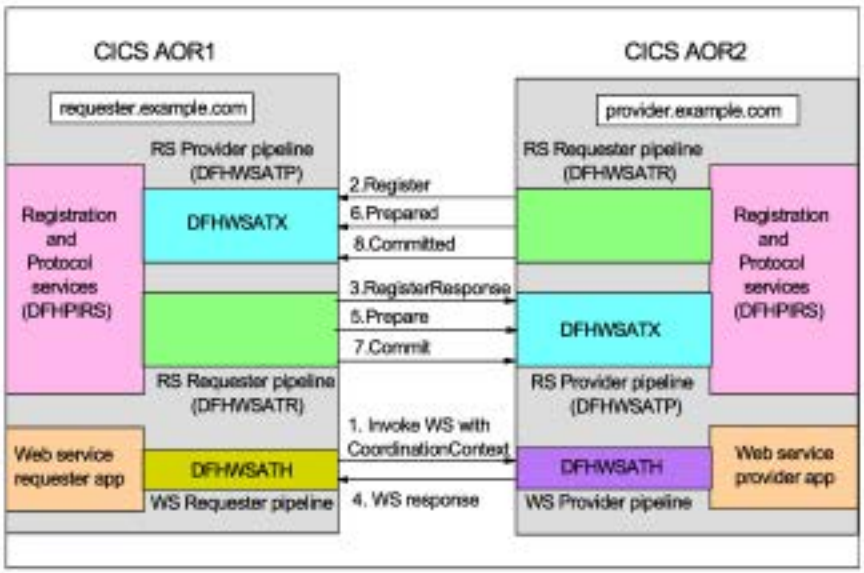


Figure 11-1 The special case: CICS to CICS

Both regions have a requester pipeline named DFHWSATR and a provider pipeline named DFHWSATP for registration and protocol processing. The DFHWSATP pipeline invokes the CICS-supplied message handler DFHWSATX as the last message handler in the pipeline.

For the workload we have shown, AOR1's DFHWSATP pipeline receives registration requests and protocol notifications, while its DFHWSATR pipeline sends registration responses and protocol instructions.

AOR2's DFHWSATP pipeline receives registration responses and protocol instructions, while its DFHWSATR pipeline sends registration requests and protocol notifications.

Note: The DFHWSATP pipeline acts as the registration endpoint for CICS.

CICS TS V3.1 provides a new resource group DFHWSAT to assist customers with setting up WS-AT support in CICS. The DFHWSAT group contains the resources shown in Table 11-1.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 450/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000450

Table 11-1 CICS supplied resource definitions for WS-AT

Resource	Resource name	Description
Pipeline	DFHWSATP	Registration services provider PIPELINE
Pipeline	DFHWSATR	Registration services requester PIPELINE
Urimap	DFHWSJPI	URIMAP used by the Registration services provider
Program	DFHPIRS	Registration and protocol services program
Program	DFHWSATH	SOAP header processing program
Program	DFHWSATR	Registration and coordination services handler program
Program	DFHWSATX	CICS message handler program

Since DFHLIST does not include the DFHWSAT group and you cannot add the DFHWSAT group to DFHLIST, specifying DFHLIST in the system initialization table GRPLIST parameter will not cause CICS to install DFHWSAT automatically during an initial start.

Figure 11-2 shows the definition of the DFHWSATP PIPELINE resource.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000451

```
OBJECT CHARACTERISTICS                                  CICS RELEASE = 0640
 CEDA  View Pipeline( DFHWSATP )
  Pipeline        : DFHWSATP
  Group           : DFHWSAT
  Description     :
  STatus          : Enabled            Enabled | Disabled
  Configfile      : /usr/lpp/cicsts/cicsts31/pipeline/configs/registrationservi
  (Mixed Case)    : ceePROV.xml
                  :
                  :
                  :
  Shelf           : /var/cicsts/
  (Mixed Case)    :
                  :
                  :
                  :
  wsdir           :
  (Mixed Case)    :
                  :
                                                  SYSID=3C3 APPLID=WAPC1JCC
```

Figure 11-2 Definition of the DFHWSATP PIPELINE resource

Note that the definition contains the fully-qualified name of the pipeline configuration file. If you install the CICS-supplied registrationservicePROV.xml configuration file in a different directory when you install CICS (as we did), then you must make a copy of the online DFHWSAT group, change the definition of the DFHWSATP pipeline, and install the modified group.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000452

Tip: If you attempt to modify the definition of the DFHWSATP pipeline in group DFHWSAT, you will get the message "`Unable to perform operation: DFHWSAT is IBM protected.`"

If you add the DFHWSAT group to your startup list and then attempt to override the definition of DFHWSATP that is provided in the DFHWSAT group by adding a modified DFHWSATP definition that appears later in the startup list, then you will get message DFHAM4892 W indicating that the install of the second group completed with errors.

If you copy only the DFHWSATP, DFHWSATR, and DFHRSURI definitions to a new group and try to let program autoinstall automatically install the definitions of the programs DFHWSATH, DFHWSATR, DFHWSATX, and DFHPIRS, you may also have problems. These four programs need access to containers which use CICS-key storage, and therefore they must run with EXECKEY(CICS) unless storage protection is turned off. You would have a problem if the model for the program you use for program autoinstall does not specify EXECKEY(CICS).

Example 11-1 shows the contents of the registrationservicePROV.xml file. This configuration file defines a pipeline that contains only one message handler program, DFHWSATX.

Example 11-1 The registrationservicePROV.xml pipeline configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <service>
    <terminal_handler>
      <handler>
        <program>DFHWSATX</program>
        <handler_parameter_list/>
      </handler>
    </terminal_handler>
  </service>
  <service_parameter_list/>
</provider_pipeline>
```

Figure 11-3 shows the definition of the DFHWSATR PIPELINE resource.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000453

```
OBJECT CHARACTERISTICS                                    CICS RELEASE = 0640
 CEDA  View PIpeline( DFHWSATR )
  PIpeline       : DFHWSATR
  Group          : DFHWSAT
  Description    :
  STatus         : Enabled            Enabled | Disabled
  Configfile     : /usr/lpp/cicsts/cicsts31/pipeline/configs/registrationserv
  (Mixed Case)   : iceREQ.xml
                 :
                 :
                 :
  SHelf          : /var/cicsts/
  (Mixed Case)   :
                 :
                 :
                 :
  Wsdir          :
  (Mixed Case)   :
                 :

                                                  SYSID=T3C1 APPLID=A6POT3C1
```

Figure 11-3 Definition of the DFHWSATR PIPELINE resource

Example 11-2 shows the contents of the registrationserviceREQ.xml file. This configuration file defines a pipeline that does not contain any message handlers.

Example 11-2 The registrationserviceREQ.xml pipeline configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
    xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
    requester.xsd">
</requester_pipeline>
```

Important: We recommend that you do not modify the registrationserviceREQ.xml pipeline configuration file or the registrationservicePROV.xml pipeline configuration file. If you modify one of these files, you might inadvertently alter the flow of the registration or protocol service messages and thereby affect the integrity of your data.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000454

Figure 11-4 shows the definition of the DFHRSURI URIMAP resource.

```
 OBJECT CHARACTERISTICS                                  CICS RELEASE = 0640
  CEDA View Urimap( DFHRSURI )
   Urimap          : DFHRSURI
   Group           : DFHWSAT
   Description     :
   Status          : Enabled            Enabled | Disabled
   USAge           : Pipeline           Server | Client | Pipeline
  UNIVERSAL RESOURCE IDENTIFICA
   SCheme          : HTTP               HTTP | HTTPS
   HOST            : *
   (Lower Case)    :
   PATH            : /cicswsat/RegistrationService
   (Mixed Case)    :
                   :
                   :
                   :
  ASSOCIATED CICS RESOURCES
   TCPipeservice   :
   Analyzer        : No                 No | Yes
   CONverter       :
   TRANsaction     : CPIH
   PROgram         :
   PIpeline        : DFHWSATP
   WEBservice      :                                          (Mixed Case)
  SECURITY ATTRIBUTES
   USEIrc          :
   Ciphers         :
```

Figure 11-4 Definition of the DFHRSURI URIMAP resource

When the CICS-supplied CWXN transaction finds that the URI in an HTTP request matches the PATH attribute of the DFHRSURI URIMAP definition, it uses the PIPELINE attribute of that definition to get the name of the PIPELINE definition that it will use to process the incoming request. As Figure 11-4 shows, this is the DFHWSATP pipeline. As we have already seen, the DFHWSATP PIPELINE definition specifies that CICS should use registrationservicePROV.xml as the pipeline configuration file. As we have also already seen, this configuration file defines a pipeline that contains only one message handler program, DFHWSATX. Thus when the URI in an HTTP request contains /cicswsat/RegistrationService, CICS invokes the DFHWSATX message handler.

This raises the question: What causes CICS to receive an HTTP request whose URI field contains /cicswsat/RegistrationService?

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000455

The answer is that the pipeline configuration file for the service requestor application (running in CICS AOR1) must specify:

- One of the CICS-provided SOAP message handlers (cics_soap_1.1_handler or cics_soap_1.2_handler)
- The mandatory invocation of the DFHWSATH header processing program to add a CoordinationContext header to the SOAP request
- A <registration_service_endpoint> element within a <service_parameter_list>

This is shown in Example 11-3

Example 11-3 Service requester pipeline configuration file which supports WS-AT

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
      requester.xsd">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>DFHWSATH</program_name>
          <namespace>
             http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </service_handler_list>
  </service>
  <service_parameter_list>
    <registration_service_endpoint>
       http://requester-example.com:3207/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</requester_pipeline>
```

The <registration_service_endpoint> element contains the address of the Registration service endpoint that runs in the requesting CICS region (AOR1). The path component of this address matches the PATH attribute defined in the DFHRSURI URIMAP resource definition of AOR1. Participant Web services should send Register requests and Prepared and Committed (or Aborted) notifications to this address.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000456

In the service requester pipeline in Example 11-3

- Since the <mandatory> element contains True, the pipeline will flow a CoordinationContext with the message.
- If you change the <mandatory> element to false or remove DFHWSATH from the pipeline, the pipeline will not flow a CoordinationContext with the message.

The pipeline configuration file (Example 11-4) for the service *provider* application must specify:

- One of the CICS-provided SOAP message handlers (cics soap 1.1 handler or cics soap 1.2 handler)
- Invocation of the DFHWSATH header processing program whenever the SOAP message contains a CoordinationContext header
- A <registration_service_endpoint> element within a <service_parameter_list>

Example 11-4 Service provider pipeline configuration file that supports WS-AT

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
      provider.xsd ">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>DFHWSATH</program_name>
          <namespace>
             http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>false</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
  <service_parameter_list>
    <registration_service_endpoint>
    http://provider.example.com:3207/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</provider_pipeline>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000457

This time the <registration_service_epr> element contains the address of the Registration service endpoint that runs in the provider CICS region. The Coordinator should send RegisterResponse messages and Prepare and Commit (or Abort) notifications to this address.

In the service provider pipeline in Example 11-4:

- The pipeline will accept flows with a CoordinationContext, and such flows will be treated as part of a WS-AT transaction
- Since the <mandatory> element contains False, the pipeline will also accept messages *without* a CoordinationContext but they will *not* be part of any WS-AT transaction
- If you change the <mandatory> element to True, the pipeline will *require* that a CoordinationContext flow with the message. A Fault will be raised if a requestor attempts to use the service without a CoordinationContext.
- If you remove DFHWSATH from the pipeline, the pipeline will raise a mustUnderstand fault when a CoordinationContext arrives with mustUnderstand set to True

We conclude this section with a few remarks on the function provided by two programs in the CICS-supplied DFHWSAT group.

DFHPIRS

DFHPIRS is the central component of CICS support for WS-AT. It provides the bulk of the function that is needed for registration and protocol processing. The actions that it can be called to service include the following: Register, RegisterResponse, Prepare, Prepared, Aborted, ReadOnly, Commit, Rollback, Committed, and Replay.

If DFHPIRS encounters an unrecoverable error, it abends with an abend code of APIO.

DFHWSATH

When included in the configuration file for a Web service *requester* pipeline, DFHWSATH controls the processing that causes a CoordinationContext to be created and added to the SOAP message before it is sent. To do this DFHWSATH calls DFHPIAT which finds the local netname, the unit-of-work ID (UOWID), and the value of the DTIMOUT attribute on the TRANSACTION definition for the transaction ID under which the requesting application is running. DFHPIAT also finds the endpoint address of the Registration service for the requesting region. From all of this information DFHPIAT creates a CoordinationContext.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000458

When included in the configuration file for a Web service *provider* pipeline, this program is invoked if the CICS-supplied SOAP message handler detects a `CoordinationContext` header in a message. When called, DFHWSATH controls the processing that extracts data from the `CoordinationContext` header.

If DFHWSATH encounters an unrecoverable error, it abends with an APIP abend, which causes the pipeline manager to terminate the processing of the current request.

11.1.2 More elaborate CICS to CICS configuration

In the simple CICS to CICS configuration shown in 11.1.1, "CICS to CICS configuration" on page 420, all the requester registration endpoints reside in the same region as the service requester application, and all the provider registration endpoints reside in the same region as the service provider application. This does not have to be the case, and a number of different configurations are possible, including the one shown in Figure 11-5.

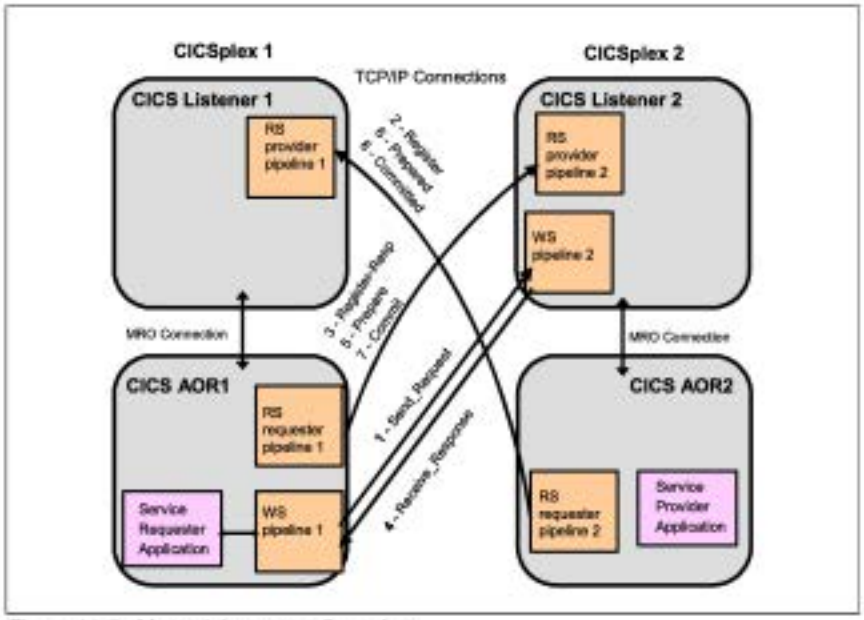


Figure 11-5 More elaborate configuration

The service requester pipeline (WS pipeline 1) must run in the same region as the service requester application that it supports. However, the service provider pipeline (WS pipeline 2) does not have to run in the same region as the service

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 459/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000459

provider application that it manages. Instead, these applications are eligible for dynamic routing as described in 3.4, "Configuring for high availability" on page 101. Similarly, Registration services provider pipelines, such as RS provider pipeline 1 and RS provider pipeline 2 in Figure 11-5, do not have to be located in the same region as the applications that they service.

In the case of RS provider pipeline 1, the requests arriving there are for registration with the service requester application running in AOR1, and protocol response messages that refer to this application's role in coordinating the atomic transaction. The RS provider pipeline 2 receives registration response messages and protocol notification messages intended for the service provider application running in AOR2.

Since eventually these messages must be processed in the same region as the service provider application, the WS-AT implementation in CICS TS makes use of *RequestStreams* to pass these messages from one region to another. RequestStreams are dependent on MRO connections when they exchange information between CICS regions.

Note: If a Registration Service provider pipeline is configured in a different region than the application that it services, then an MRO connection is required between the CICS regions.

The use of RequestStreams means that individual Registration services provider pipelines can be used by different applications, and those applications do not have to reside in the same region. A set of cloned AORs can share a common Registration services provider pipeline that is located in a specific WS-AT CICS region. Work can then be distributed across the AORs and each of the WS-AT messages, arriving at the Registration services provider pipeline, contains sufficient information to allow it to be forwarded to the correct AOR.

Tip: A set of cloned AORs can share a common RS provider pipeline located in a specific WS-AT region, thus reducing the number of pipeline configurations required. In this configuration, it is recommended that you do not run applications in the specific WS-AT region and that you create clones for improved failover and availability.

Figure 11-5 shows the Registration services requester pipelines are in the same region as the applications that they service. RS requester pipeline 1 services the service requester application in AOR1, and RS requester pipeline 2 services the service provider application in AOR2. This configuration cannot be changed because CICS requires that each WS-AT message must be dispatched from the same region as the application that it relates to. This means that each AOR in a

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000460

cloned set of AORs used to support a common Web services workload, must have it's own Registration services requester pipeline configured.

Note: A Registration services requester pipeline must be configured in each AOR that hosts applications which participate in atomic transactions.

11.2 Enabling atomic transactions in WebSphere

WebSphere Application Server Version 6 implements the WS-AT specification. A J2EE application programmer demarcates a global transaction in a program by using the Java Transaction API (JTA) `UserTransaction` interface as shown in Example 11-5.

Example 11-5 Demarcating a global transaction using the JTA

```
UserTransaction userTransaction = null;
try {
      InitialContext context = new InitialContext();
      userTransaction = (UserTransaction)
                              context.lookup("java:comp/UserTransaction");
      userTransaction.begin();

      // insert record into database
        .
        .
        .
      // commit
      userTransaction.commit();
    } catch (java.rmi.RemoteException re) {
        try {
           userTransaction.rollback();
        }
        .....
     }
```

If a Web service request is made by an application component running under a global transaction, WebSphere Application Server implicitly propagates a `CoordinationContext` to the target Web service if the appropriate application deployment descriptors have been specified.

If WebSphere Application Server is the system hosting the target endpoint for a Web service request that contains a WS-AT `CoordinationContext`, WebSphere automatically establishes a subordinate JTA transaction in the target run-time environment that becomes the transactional context under which the target Web service application executes.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net

Page 461/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000461

Application developers do not have to explicitly register WS-AT participants. The WebSphere Application Server runtime takes responsibility for the registration of WS-AT participants. At transaction completion time, all WS-AT participants are atomically coordinated by the WebSphere Application Server transaction service.

There are no specific development tasks required for Web service applications to take advantage of WS-AT; however, there are some application deployment descriptors that have to be set appropriately.

- In a Web module that invokes a Web service, specify Send Web Services Atomic Transaction on requests to propagate the transaction to the target Web service.

  See "Change the deployment descriptor" on page 450 for information on how we enabled our Web application to send WS-AT SOAP headers in requests to a CICS service provider application.

- In a Web module that implements a Web service, specify Execute using Web Services Atomic Transaction on incoming requests to run under a received client transaction context.
- In an EJB module that invokes a Web service, specify Use Web Services Atomic Transaction to propagate the EJB transaction to the target Web service.
- In an EJB module, bean methods must be specified with transaction type Required, which is the default, to participate in a global atomic transaction.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000462

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000463

12

Transaction scenarios

In this chapter we show different scenarios that demonstrate how you can synchronize resource updates using the WS-Atomic Transaction support in CICS and WebSphere Application Server.

We start with an explanation of the scenarios that we set out to test, and how we prepared the system and the settings that we used for the configuration of our system.

We then cover in detail the changes that we made to the CICS pipeline configuration and WebSphere Application Server to enable Web services transactional integration.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000464

12.1 Introduction to our scenarios

Figure 12-1 shows three different atomic transaction scenarios that we considered testing.

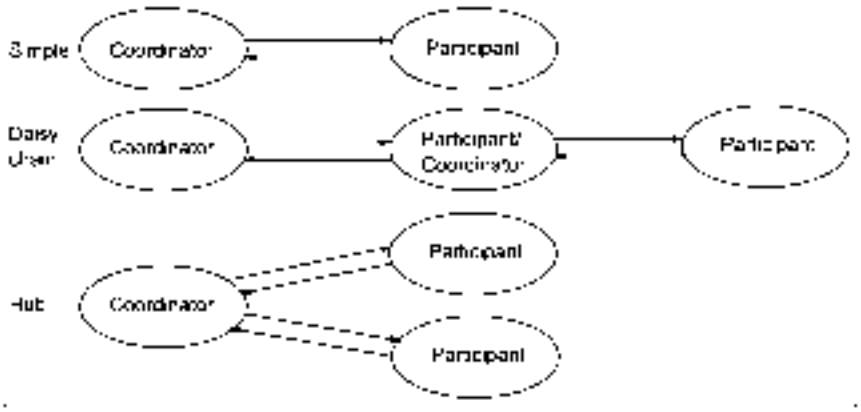


Figure 12-1 Three possible atomic transaction scenarios

The scenario at the top is the simple case where a coordinator controls a single participant. Both parties may make recoverable updates. Another possibility is that only the participant makes recoverable updates. A third possibility is that the participant does nothing recoverable; in this case it will send a ReadOnly notification when it is coordinated.

The second scenario is often referred to as a *daisy chain*. Here there is a primary (or root) coordinator that invokes a Web service, and the invoked Web service then invokes a second Web service. The online atomic transaction is controlled by the primary coordinator. The middle system takes on the role of a coordinator *and* the role of a participant at different times in its life cycle. When the primary coordinator instructs it during transaction termination, this system acts as a participant. However, before it responds to the primary coordinator, it then takes on the role of coordinator of its own participant.

The third scenario is a *hub* configuration. A single coordinator invokes one Web service and then another. It then coordinates them together.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000465

You can probably think of many other scenarios. In this chapter we more closely look at two of these scenarios:

- The simple scenario in which WebSphere Application Server V6.0 is the coordinator and CICS TS V3.1 is the participant. See "The simple atomic transaction scenario" on page 439.
- The daisy chain scenario in which CICS TS V3.1 is both a coordinator and a participant. See 12.3, "The daisy chain atomic transaction scenario" on page 482.

12.1.1 Software checklist

Table 12-1 shows the software we used in the scenarios described in this chapter.

Table 12-1 Software used in the atomic transaction scenarios

Windows	z/OS
Windows 2000 SP4	z/OS V1.6
IBM WebSphere Application Server - ND V6.0.0.2	CICS Transaction Server V3.1
Internet Explorer V6.0	
DB2 V8.1.7.445	
User-supplied programs: - CatalogAtomic.ear A modified service requester application used for the WS-AT scenarios - DispatchAtomic.ear A modified service provider application used for the WS-AT scenarios	User-supplied programs: - SNIFFER (message handler program written in COBOL) - WSATHND (header processing program written in C)

Important: You should install the fix for APAR PK16509 if you are using a later version of WebSphere than IBM WebSphere Application Server - Network Deployment V6.0.0.2.

12.1.2 Definition checklist

Table 12-2 shows the definitions we used in the scenarios described in this chapter.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 466/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000466

Table 12-2 Definitions used in the atomic transaction scenarios

Value	CICS TS	WebSphere Application Server
IP name	mvsg3.mop.ibm.com	cam21-pc1[illegible].mop.ibm.com
IP address	9.[illegible]0.193.167	9.[illegible]0.199.238
Job name	CWST3C1	
APPLID	A6POT3C1	
SIT parameter	SEC=NO (see APAR PK10840)	
TCPIPSERVICE definition	T3C1	
PORT attribute on T3C1 TCPIPSERVICE definition	15351	
FILE definition for sample catalog VSAM file	EXMPCAT	
RECOVERY attribute on EXMPCAT FILE definition	BACKOUTONLY	
Web service provider PIPELINE definition	PIPE1	
CONFIGFILE attribute on PIPE1 PIPELINE definition	/CICSTS31/CWS1/config/ITSO_7206_wsat_soap11provider.xml	
Web service requester PIPELINE definition	PIPE2	
CONFIGFILE attribute on PIPE2 PIPELINE definition	/CICSTS31/CWS1/config/ITSO_7206_wsat_soap11requester.xml	
RDO group containing copy of DFHWSAT group	CTS3100	
Registration Service provider PIPELINE	DFHWSATP	
CONFIGFILE attribute on DFHWSATP PIPELINE definition	/CICSTS31/CWS1/config/ITSO_7206_wsat_registrationservicePROV.xml	
Registration Service requester PIPELINE	DFHWSATR	

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000467

Value	CICS TS	WebSphere Application Server
CONFIGFILE attribute on DFHWSATR PIPELINE definition	/CIWS/T3C1/config /ITSO_7206_wsat_registrationserviceREQ.xml	

Important: As we write this book, APAR PK10849 is open. It reports that if you attempt to use WS-AT in a CICS region that is running with the SIT parameter SEC=YES, then during the RegisterResponse step of coordination RACF will issue messages ICH408I and IRR012I and CICS will issue message DFHPI0002. Therefore, we ran our region with SEC=NO.

12.2 The simple atomic transaction scenario

The sample Catalog application provided with CICS TS V3.1 provides three Web services:

- inquireSingle
- inquireCatalog
- placeOrder

Only the placeOrder Web service *updates* the VSAM file that contains the information about the company's products. Therefore, we naturally decided that the service requester running in WebSphere Application Server should invoke the placeOrder Web service in our WS-AT scenario.

We modified the service requester application (see "Installing the service requester" on page 87) to create a global transaction and to update a DB2 table. We call the new service requester application *AtomicClient*.

Note: Rather than using JDBC™ to update a DB2 table, we could have chosen to use another J2EE connector such as the JCA or JMS. Updates to resources accessed by these connectors can be synchronized with Web service requests in the same way.

To be more specific, we create the table ITSO.ORDER in DB2. Before calling the placeOrder Web service, the AtomicClient inserts a row in this table so that we have a log of all of the orders placed through our application. We can now have a global transaction that updates two resources: a DB2 table in the Windows environment and a VSAM file in the z/OS environment. Figure 12-2 shows the

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 468/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000468

sequence of events in AtomicClient as it begins a global transaction, updates a DB2 database, calls the placeOrder Web service, and then either commits or rolls back the updates.

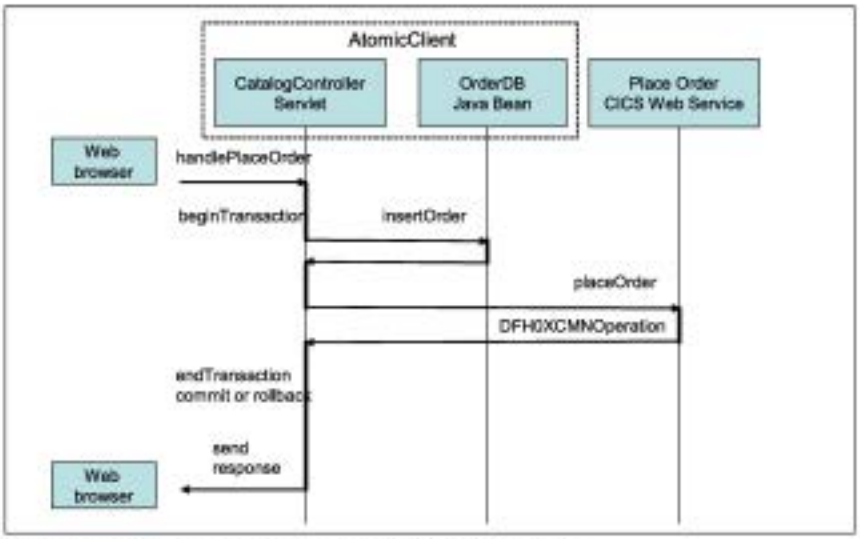


Figure 12-2 Simple atomic transaction sequence of events

Figure 12-3 shows a more global view of the sequence of events:

- The user uses his Web browser to invoke the AtomicClient that runs in WebSphere Application Server.
- The AtomicClient updates the ITSO.ORDER table in DB2.
- WebSphere Application Server sends the insertOrder SOAP message containing the order to CICS.
- CICS uses Web service provider PIPELINE definition PIPE1 to process the SOAP message. PIPE1 contains our SNIFFER program and the CICS-supplied SOAP 1.1 message handler DFHPISN1.
- DFHPISN1 links to the CICS-supplied header processing program DFHWSATH.
- CICS converts the SOAP message to a COMMAREA for the sample catalog manager program, DFH0XCMN.
- DFH0XCMN passes the data in the COMMAREA to the sample catalog program DFH0XVDS, which updates the recoverable VSAM file.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 469/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000469

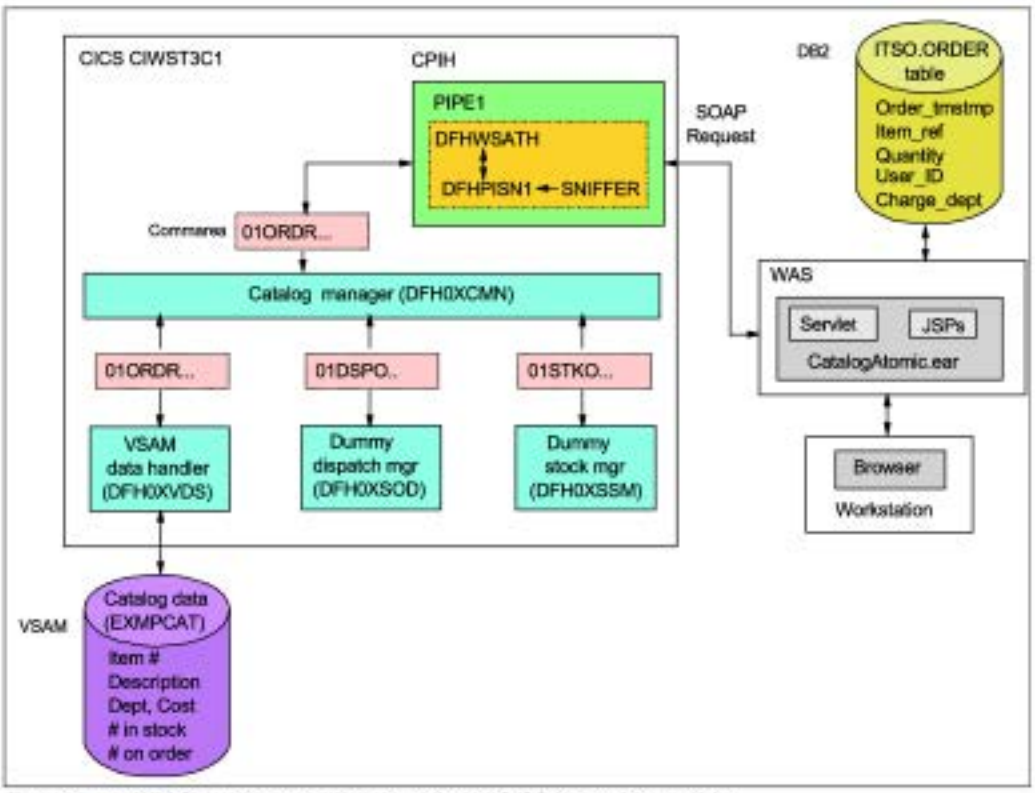


Figure 12-3 WebSphere as service requester and CICS as service provider

In the following sections we describe how we set up CICS for this scenario, how we created the AtomicClient and the ITSO.ORDER table, and the results of testing this scenario.

12.2.1 Setting up CICS for the simple scenario

To set up CICS for this scenario we performed the following steps:

- Set the value of the SEC system initialization table parameter to NO; see 12.1.2, "Definition checklist" on page 437.
- Changed the value of the RECOVERY attribute of the EXMPCAT FILE definition to BACKOUTONLY so that the file is recoverable. (The EXMPCAT FILE definition defines the VSAM file that contains the catalog data for the sample application).

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
Online/Notary.net
Page 470/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000470

- Edited PIPE1's configuration file

  /CIWS/T3C1/config/ITSO_7206_wsat_soap11provider.xml

  so that it contains the XML shown in Example 12-1. This XML contains:

  - The same XML as shown in Example 11-4 on page 428, except that we changed the address in the registration_service_endpoint element to be specific to the CICS region CIWST3C1
  - XML to add the SNIFFER message handler

Example 12-1 PIPE1: /CIWS/T3C1/config/ITSO_7206_wsat_soap11provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
      provider.xsd ">
  <service>
    <service_handler_list>
      <handler>
        <program>SNIFFER</program>
        <handler_parameter_list/>
      </handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <header_program>
          <program_name>DFHWS4IH</program_name>
          <namespace>
             http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>false</mandatory>
        </header_program>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
  <service_parameter_list>
    <registration_service_endpoint>
     http://MVS63.mop.ibm.com:15301/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</provider_pipeline>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000471

Tip: For readability Example 12-1 shows the `registration_service_endpoint` element split across three lines. However, during our testing we found that we had to place its start tag, contents, and end tag on the *same* line.

- Copied the group DFHWSAT to the group CTS310C and added the group CTS310C to our startup list:

```
CEDA COPY GROUP(DFHWSAT) TO(CTS310C)
CEDA ADD GROUP(CTS310C) LIST(LISTT3C1)
```

- Edited DFHWSATP's configuration file

  /CIWS/T3C1/config/ITSO_7206_wsat_registrationservicePROV.xml

  so that it contained the XML shown in Example 12-2. This XML contains:

  - The same XML as shown in Example 11-1 on page 424
  - XML to add the WSATHND header processing program (see "How we monitored the exchange of WS-AT messages" on page 444)

Example 12-2 DFHWSATP: /CIWS/.../ITSO_7206_wsat_registrationservicePROV.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <service>
    <service_handler_list>
      <cics_soap_1.2_handler>
        <headerprogram>
          <program_name>WSATHND</program_name>
          <namespace>*</namespace>
          <localname>wsatHeader</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.2_handler>
    </service_handler_list>
    <terminal_handler>
      <handler>
        <program>DFHWSATX</program>
        <handler_parameter_list/>
      </handler>
    </terminal_handler>
  </service>
  <service_parameter_list/>
</provider_pipeline>
```


Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 472/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000472

- Edited DFHWSATR's configuration file

  /CIWS/T3C1/config/ITSO_7206_wsat_registrationserviceREQ.xml

  so that it contained the XML shown in Example 12-3. This XML contains

  - The same XML shown in Example 11-2 on page 425
  - XML to add the WSATHND header processing program

Example 12-3 DFHWSATR: /CIWS/.../ITSO_7206_wsat_registrationserviceREQ.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
    xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
    requester.xsd">
  <service>
    <service_handler_list>
      <cics_soap_1.2_handler>
        <headerprogram>
          <program_name>WSATHND</program_name>
          <namespace></namespace>
          <localname>wsatHeader</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.2_handler>
    </service_handler_list>
  </service>
</requester_pipeline>
```

How we monitored the exchange of WS-AT messages

The user-written WSATHND header processing program writes messages to the CESO transient data queue, which normally is an extrapartition queue with ODname CEEOUT. We used WSATHND to monitor the exchange of registration and protocol service messages between WebSphere Application Server and CICS.

The message output of WSATHND depends on the values the programmer has assigned to the two variables MESSAGES_ON and FULL_MESSAGES_ON.

1. If MESSAGES_ON is set to 1, the program writes the following message:

   WSAT: REACHED HANDLER - function

   where function is the contents of the DFHFUNCTION container (RECEIVE-REQUEST, SEND-RESPONSE, SEND-REQUEST, RECEIVE_RESPONSE, PROCESS_REQUEST, NO_RESPONSE, or HANDLER-ERROR)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000473

When the function is NO-RESPONSE, then the handler is being invoked after processing a request, when there is no response to be processed.

2. If the function is not NO-RESPONSE, then:
   - If FULL_MESSAGES_ON is set to 1, WSATHND writes the following message:

     WSAT: contents of the DFHREQUEST container
   - If MESSAGES_ON is set to 1, it will write the following message:

     WSAT: ACTION: action

     where action has one of the following values: Register, RegisterResponse, Prepare, Prepared, Commit, Committed, ReadOnly, Abort, Aborted, Rollback

Thus, WSATHND provides a way to see the registration and protocol service messages that are being sent between the service requester and the service provider as they happen. We found it easier to use this program than to take a TCP/IP trace or to search for these messages in a CICS auxiliary trace. The full WSATHND program is shown in A.7, "Sample header processing program - WSATHND" on page 539.

12.2.2 Creating the AtomicClient and ITSO.ORDER table

To create the AtomicClient and the ITSO.ORDER table, we did the following:

- Created the OrderBean JavaBean that represents the order.
- Created the OrderDB JavaBean that inserts the order into the DB2 database.
- Changed the handlePlaceOrder method of the CatalogController servlet so that it creates an OrderBean and then calls OrderDB to insert it into the DB2 database as part of a global transaction.
- Changed the deployment descriptor of the Web module CatalogAtomicWeb. This is the module that invokes the Web Service.
- Created the ITSO.ORDER table.
- Created a data source.

We explain each step in detail in the following sections.

Create the OrderBean JavaBean

The OrderBean JavaBean represents the order; it has all the fields from the PlaceOrder JavaServer™ Page (JSP™) plus a timestamp that records the time when the Web browser user placed the order. Example 12-4 shows the code for the OrderBean JavaBean.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000474

Example 12-4 The OrderBean JavaBean

```
package itso.mop.objects;

import java.sql.Timestamp;

public class OrderBean {

   private Timestamp orderTmstmp;
   private short itemRef;
   private short quantity;
   private String userId;
   private String chargeDept;

// getters and setters
...
}
```

Create the OrderDB JavaBean

Example 12-5 shows the code for the OrderDB JavaBean, which inserts the order into the DB2 table.

Example 12-5 The OrderDB JavaBean

```
package itso.mop.db;

import  itso.mop.objects.OrderBean;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class OrderDB {

   private static final String _JNDI_NAME = "jdbc/CACHEDS";

   public  int insertOrder(OrderBean order){
      Connection con = getConnection();
      PreparedStatement pr = null;
      int rs  = 0;
      try {
         pr = con.prepareStatement("INSERT INTO ITSO.ORDER (ORDER_TMSTMP,
ITEM_REF, QUANTITY, USER_ID, CHARGE_DEPT) VALUES(?,?,?,?,?)");
         pr.setTimestamp(1, order.getOrderTmstmp());
         pr.setInt(2, order.getItemRef());
         pr.setInt(3, order.getQuantity());
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000475

```
            pm.setString(4, order.getUserId());
            pm.setString(5, order.getChargeCost());

            rs = pm.executeUpdate();

            System.out.println("OrderDB.insertOrder() - Inserted the order in the
database!!!");
         } catch (SQLException e) {
            System.out.println("OrderDB.insertOrder()  Exception inserting the
order in the database!!");
            e.printStackTrace(System.err);
            rs = 0;
         } finally {
            try {
               if (pm != null)
                  pm.close();
               if (con != null)
                  con.close();
            } catch (Exception e) {
               e.printStackTrace(System.err);
            }
         }
         return rs;
      }

      private Connection getConnection() {
         Connection con = null;
         try {
            InitialContext ic = new InitialContext();
            DataSource ds = (DataSource) ic.lookup("JNDI_NAME");
            con = ds.getConnection();
         } catch (NamingException e) {
            e.printStackTrace(System.err);
         } catch (SQLException e) {
            e.printStackTrace(System.err);
         }
         return con;
      }
   }
```

Change the handlePlaceOrder method

In the Catalog application the CatalogController servlet handles all the requests from the Web browser. In particular, the handlePlaceOrder method of this servlet handles requests from the Web browser to place an order. We made the following changes to this method:

1. Created the OrderBean and populated all of its fields

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000476

2. Created the transaction (UserTransaction) and then began the transaction
3. Inserted the order into the DB2 database using the OrderDB JavaBean
4. Called the CICS Web service
5. Coded the method to throw a RemoteException when the "ROLLBACK" user ID enters the order (for testing transaction rollback)
6. Committed the transaction
7. Rolled back the transaction in the case of an Exception

Note: The rollback logic was added to the application for testing purposes only; normal applications would not be expected to behave in this way.

Example 12-6 shows the code for the `handlePlaceOrder` method.

Example 12-6 The handlePlaceOrder method of the CatalogController servlet

```
private void handlePlaceOrder(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
   OrderDetails orderDetails = new OrderDetails();
   try
   {
      // Retrieve the data from the submitted form
      orderDetails.setItemRefNumber(
short.parseShort(request.getParameter("itemRef")));
      orderDetails.setQuantityRequired(Short.parseShort(
         request.getParameter("quantity")));
      orderDetails.setUserId(request.getParameter("userName"));
      orderDetails.setChargeDepartment(request.getParameter("deptName"));
   }
   catch(Exception e)
   {
      handleError(request, response, "Input data format incorrect, please try
again");
      return;
   }

   UserTransaction userTransaction = null;
   try
   {
      //1. Create the order bean and populate all of its fields
      System.out.println("CatalogController.handlePlaceOrder() - creating the
order");
      OrderBean order = new OrderBean();
      order.setOrderTmstmp(new Timestamp(Calendar.getInstance().
```


Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000477

```
            getTime().getTime()));
        order.setItemRef(orderDetails.getItemRefNumber());
        order.setQuantity(orderDetails.getQuantityRequired());
        order.setUserId(orderDetails.getUserId());
        order.setChargeDept(orderDetails.getChargeDepartment());

        // 2. Create and begin the Transaction
        InitialContext context = new InitialContext();
        userTransaction = (UserTransaction)
                        context.lookup("java:comp/UserTransaction");
        System.out.println("CatalogController.handlePlaceOrder() - beginning the
transaction");
        userTransaction.begin();

        // 3. Insert the order into the database
        System.out.println("CatalogController.handlePlaceOrder() - inserting the
order in the database");
        OrderDB orderDB   new OrderDB();
        orderDB.insertOrder(order);

        // 4. Make the Web service call to CICS
        System.out.println("CatalogController.handlePlaceOrder() - calling the
CICS web service");
        orderDetails.populateResponse(
getOrderProxy(request).DFHZXCMNOperation(orderDetails.getRequestProgramIdentifier
()) );
        System.out.println("CatalogController.handlePlaceOrder() - response back
from the CICS web service");
        // 5. Throw exception if user ID ROLLBACK
        if(order.getUserId().equalsIgnoreCase("ROLLBACK"))
        {
           System.out.println("CatalogController.handlePlaceOrder() - simulating
the RemoteException");
           throw new RemoteException("Throwing the RemoteException");
        }
        // 6. Commit the transaction
        System.out.println("CatalogController.handlePlaceOrder() - commit the
transaction");
        userTransaction.commit();
        System.out.println("CatalogController.handlePlaceOrder()   after
commit");
     }
        // 7. Rollback in case of exception
     catch (Exception e)
     {
        try {
           System.out.println("CatalogController.handlePlaceOrder() -
exception thrown");
           userTransaction.rollback();
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000478

```
      } catch (Exception e1) {
         System.out.println("CatalogController.handlePlaceOrder() - Exception
when rollingback the transaction");
         e1.printStackTrace();
      }

      if (e.getMessage().startsWith("java.net.ConnectException"))
      {
         String errorMessage = "Unable to connect to service endpoint: "+
                        getOrderProxy(request).getEndpoint()+
                        "<BR> Please ensure service is running.";
         handleError(request, response, errorMessage);
         return;
      }
      else
      {
         e.printStackTrace();
         handleError(request, response, "An Error occured calling the service:
"+e.getMessage());
         return;
      }
   }
   // Call the response page setting the OrderDetails bean on the request
   RequestDispatcher dispatcher =
request.getRequestDispatcher(orderResponsePage);
   request.setAttribute("orderDetails",orderDetails);
   dispatcher.forward(request,response);
}
```

Note: Since the service requester application is a Web application (as opposed to an enterprise bean) we used a bean-managed transaction. For an enterprise bean, it is generally recommended to use container-managed transactions.

Change the deployment descriptor

The deployment descriptor of a J2EE application is used to specify whether the application component, if it makes any Web service requests, expects any transaction context to be propagated with the Web service requests (in accordance with the WebSphere WS-AT support).

To activate the WS-AT support:

- We imported the client application archive CatalogAtomic.ear into RAD. We then expanded the **Dynamic Web Project** (CatalogAtomicWeb) in the Project Explorer and opened (double-clicked) the **Deployment Descriptor** (Figure 12-4).

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 479/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000479

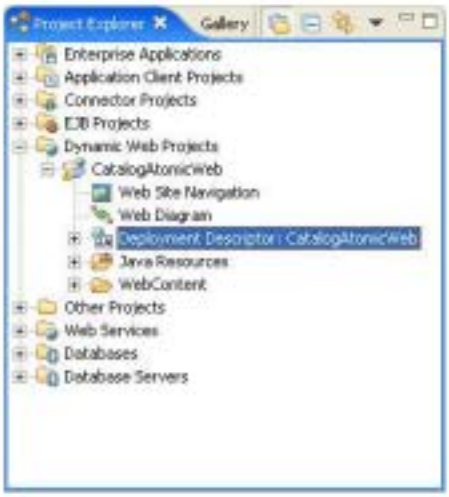


Figure 12-4 Project Explorer for CatalogAtomicWeb application

- In the Deployment Descriptor we clicked the **Servlets** tab and selected the **CatalogController** servlet. Then we scrolled down to Global Transaction and selected **Send Web Services Atomic Transactions on requests** (Figure 12-5).

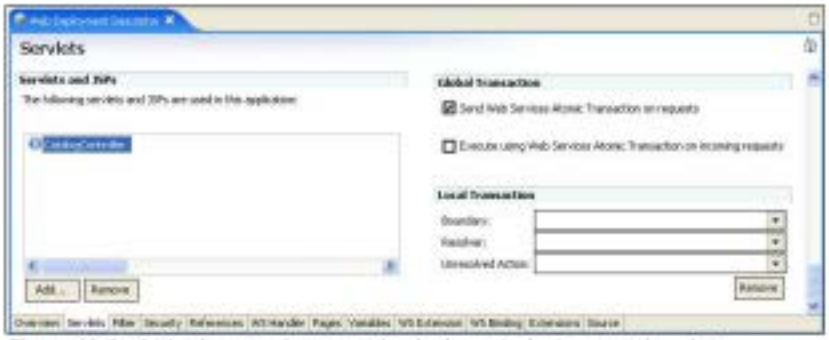


Figure 12-5 Activating atomic transaction in the web deployment descriptor

- We saved and closed the file.

Note: A similar deployment descriptor attribute **Use Web Services Atomic Transaction** can be used for enterprise beans.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 480/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000480

Create the ITSO.ORDER table

We created the ITSO.ORDER table to record all the orders that the Web browser user sends to the Catalog application. Example 12-7 shows the script we used to create the database and the table.

Example 12-7 Creation of ITSOWS database and ITSO.ORDER table

```
   CONNECT RESET;
-- Create database ITSOWS and schema ITSO
CREATE DATABASE ITSOWS;
CONNECT TO ITSOWS;
CREATE SCHEMA ITSO;

-- Table definitions for Order
CREATE TABLE ITSO.ORDER
  (ORDER_TMSTMP TIMESTAMP  NOT NULL ,
   ITEM_REF INTEGER  NOT NULL ,
   QUANTITY INTEGER  NOT NULL ,
   USER_ID CHARACTER (20)  NOT NULL ,
   CHARGE_DEPT CHARACTER (20)  NOT NULL  ,
   CONSTRAINT ORDERKEY PRIMARY KEY ( ORDER_TMSTMP)  ) ;
```

The ITSO.ORDER table has a column for every field in the Catalog Place Order JSP. Also, there is a TimeStamp column that is used as a unique key for the table. Figure 12-6 shows the ITSOWS database in the Control Center.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000481

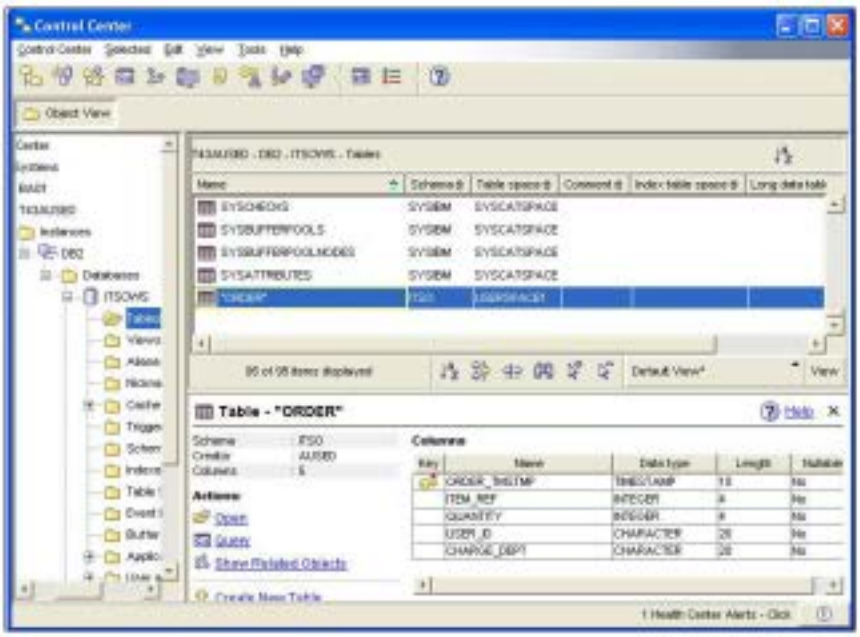


Figure 12-6 The ITSOWS database in the Control Center

Create a WebSphere data source

Next we configured the data source in WebSphere through the Administrative Console.

- We opened the Admin console by pointing a Web browser at:

  `http://cam21-pc11:9060/admin`

- In the main window, we opened **Resources** and then clicked **JDBC Providers** (Figure 12-7).

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 482/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000482

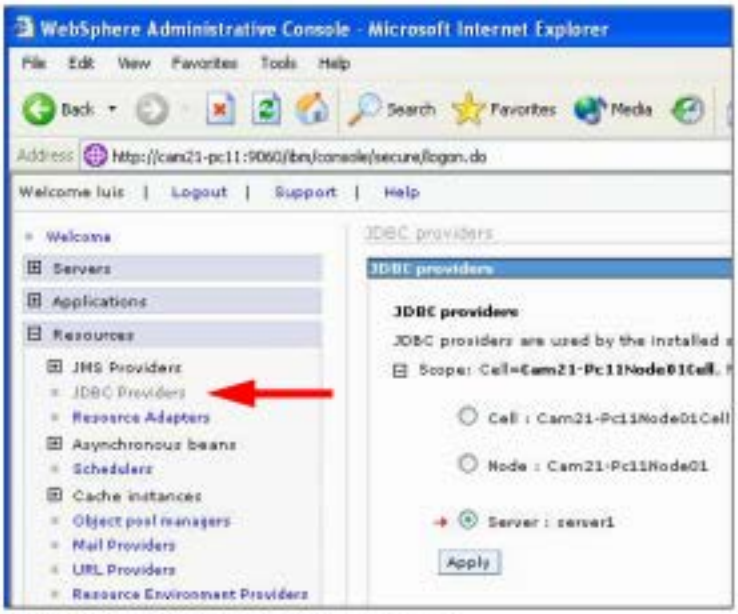


Figure 12-7 Admin console - JDBC providers

- Because we are running a single server, we chose the Server scope.
- To create a new JDBC provider we clicked **New**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 483/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000483

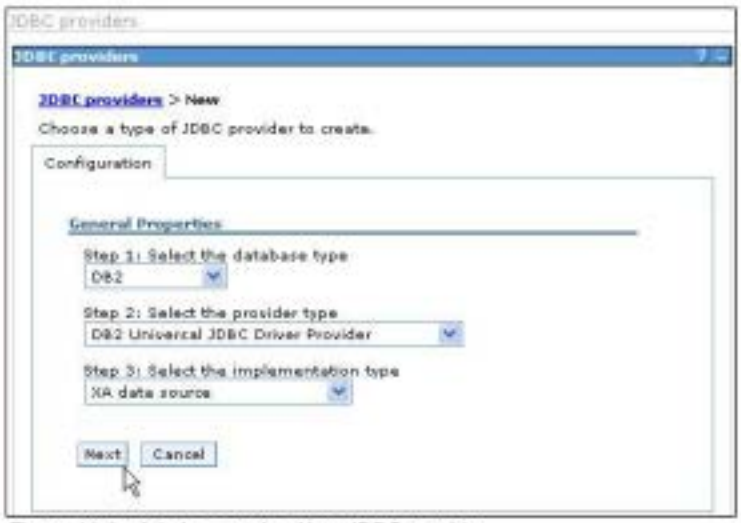


Figure 12-8 Admin console - New JDBC provider

- In the new JDBC provider window (Figure 12-8), we selected:
  - DB2 as database type
  - DB2 Universal JDBC Driver Provider as provider type
  - XA data source as the implementation type

  We clicked **Next**, and we were then presented with the window shown in Figure 12-9.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 484/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000484



Figure 12-9 Admin console - New JDBC provider 2

- We provided a name for the the JDBC provider, in this instance:

  `ITSO WebServices DB2 JDBC Driver Provider (XA)`

- We noted that WebSphere offers the Implementation class name COM.ibm.db2.jdbc.DB2XADataSource. This is the class of the DB2 driver that has two-phase commit capability. Since this class is not in the default

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 485/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000485

class path provided by WebSphere, we added to the end of the class path the following:

```
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2java.zip
```

- We clicked **OK** and saved the configuration changes.

Now we have the new JDBC provider defined and we can see it in the list of JDBC providers.

Define the DB2UNIVERSAL_JDBC_DRIVER_PATH variable

We defined the DB2UNIVERSAL_JDBC_DRIVER_PATH WebSphere variable as follows:

- In the Admin console main window we clicked **Environment** and then clicked **WebSphere Variables** as shown in Figure 12-10.



Figure 12-10 Admin console - WebSphere variables window

- We set the value of the DB2UNIVERSAL_JDBC_DRIVER_PATH variable to the default JDBC driver location C:\Program Files\IBM\SQLLIB\java (Figure 12-11).

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 486/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000486



Figure 12-11 Admin console - Defining the DB2 driver path variable

Create the J2C Authentication data

Before creating the data source, we needed to create the J2EE Connector Architecture (J2C) Authentication data in JAAS Configuration.

- In the Admin console main window we selected **Security** and then **Global Security**.
- In the Global security page we clicked **JAAS Configuration under Authentication**, and selected **J2C Authentication data**.
- In the **J2EE Connector Architecture (J2C) authentication data entries** page we clicked **New**. The window shown in Figure 12-12 was displayed.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000487



Figure 12-12 Admin console - Define J2C authentication alias

- In the General Properties we entered the following values:
  - DB2user as Alias
  - Administrator as User ID
  - User password as Password
- We clicked **OK** and saved the configuration changes.

Configure the data source for the JDBC provider

Next we configured the data source for the JDBC provider.

- In the Admin console main window we selected **Resources** and then **JDBC Providers**.
- In the JDBC providers window, we clicked the new provider that we defined in "Create a WebSphere data source" on page 453:

  `ITSO WebServices DB2 JDBC Driver Provider (XA)`

- In **Additional Properties** we clicked **Data sources**, then clicked **New**. We were presented with the window shown in Figure 12-13.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
Online/Notary.net
Page 488/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000488

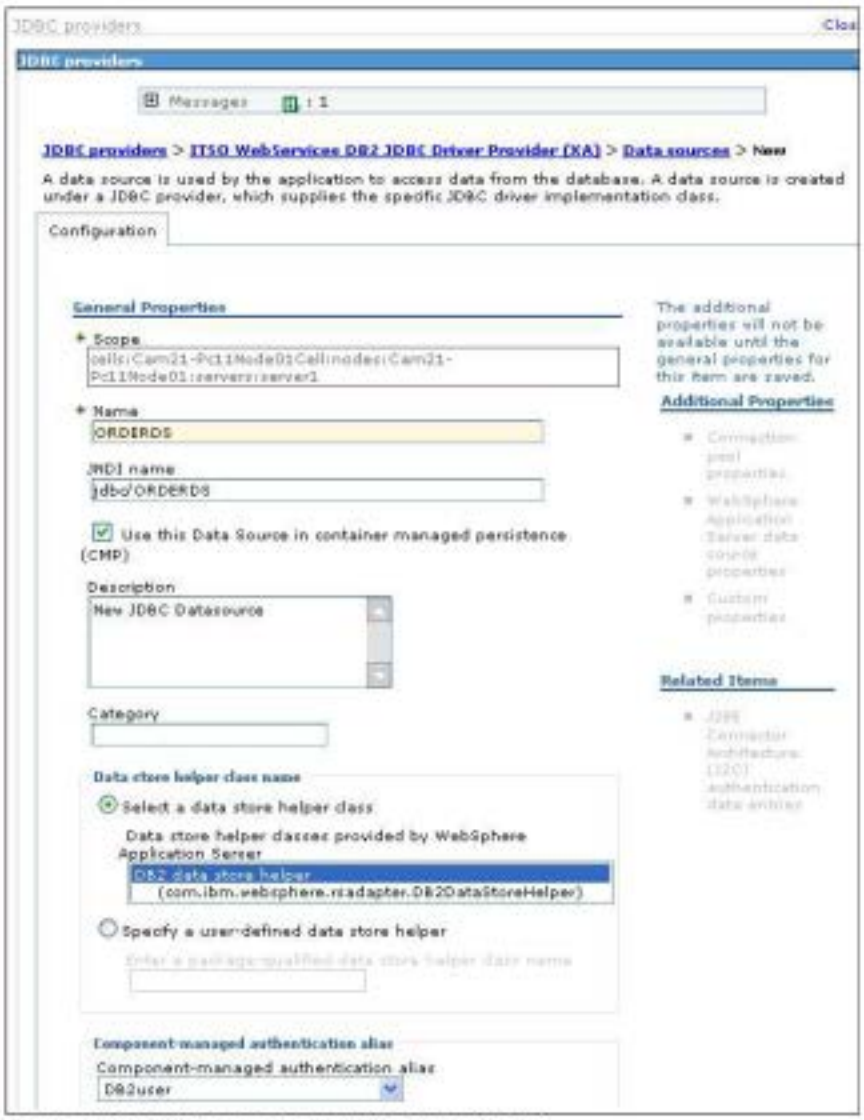


Figure 12-13 Admin console - New Data source (1 of 2)

- In the General Properties we entered the following values:
  - ORDERDS as the Data source name.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 489/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000489

- jdbc/ORDERDS as the JNDI name.
- DB2user as the Component-managed authentication alias.

- We then scrolled down to the bottom of the window and entered ITSOWS as the Database name (Figure 12-14).

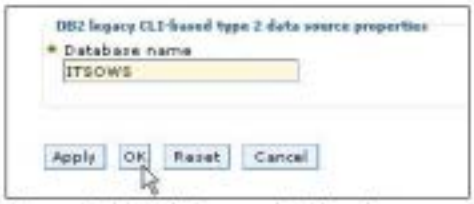


Figure 12-14 Admin console - New data source (2 of 2)

- We clicked **OK** and then saved the changes.

12.2.3 Testing the simple scenario

We performed two tests of the simple scenario:

- Normal transaction termination
- Abnormal transaction termination (see "Simple scenario: abnormal transaction termination" on page 479)

Simple scenario: normal transaction termination

In this section, we explain how we ran the AtomicClient application and then show the registration and protocol service messages that are exchanged between WebSphere Application Server and CICS during the normal termination of the atomic transaction.

To run the scenario we followed these steps:

- We opened the welcome window of the Catalog application using the URL:

  `http://cam21-pc11:9080/CatalogAtomicWeb/Welcome.jsp`

  We were presented with the window shown in Figure 12-15.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000490

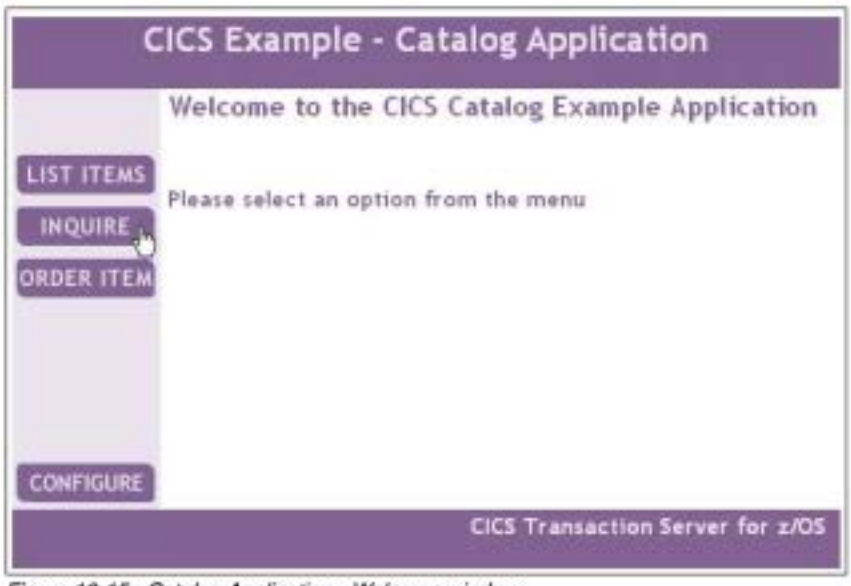


Figure 12-15 Catalog Application - Welcome window

- We clicked **INQUIRE**.
- In the Inquire Single window we used the Item Reference Number default value of 0010 and clicked **SUBMIT**. The Web service request was sent to CICS and we were presented with the results of the inquiry as shown in Figure 12-16.



Figure 12-16 Catalog Application - Inquire single

- We noted that the number of items in stock was 76. This value is taken from the CICS VSAM file.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 491/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000491

- We clicked **SUBMIT** to go to the Enter Order Details window shown in Figure 12-17.



Figure 12-17 Catalog Application - Enter order details

- In the Enter Order Details window we provided a User Name and a Department Name and clicked **SUBMIT**.
- After the CICS Web service processed the order, we got a response telling us that the order was successfully placed (Figure 12-18).



Figure 12-18 Catalog Application - Order placed response

In the WebSphere server log we see trace entries, generated by the Catalog Controller servlet, which show that the transaction was successfully committed (Example 12-8).

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 492/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000492

Example 12-8 WebSphere server log with successful Place Order

```
CatalogContoller:doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - commit the transaction
CatalogController.handlePlaceOrder() - after commit
```

- Next we checked the same item number through the Inquire Single service and verified that the stock level decreased by one item (Figure 12-19).

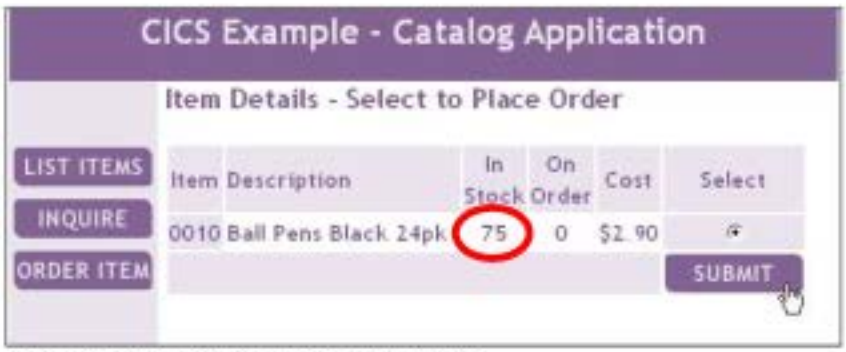


Figure 12-19 Catalog Application - Inquire single

- We opened a DB2 Control Center and issued the SQL command:

  ```
  SELECT * FROM ITSO.ORDER
  ```

  This lists all of the records in our ITSO.ORDER table. Figure 12-20 shows that our new record is in the table.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000493

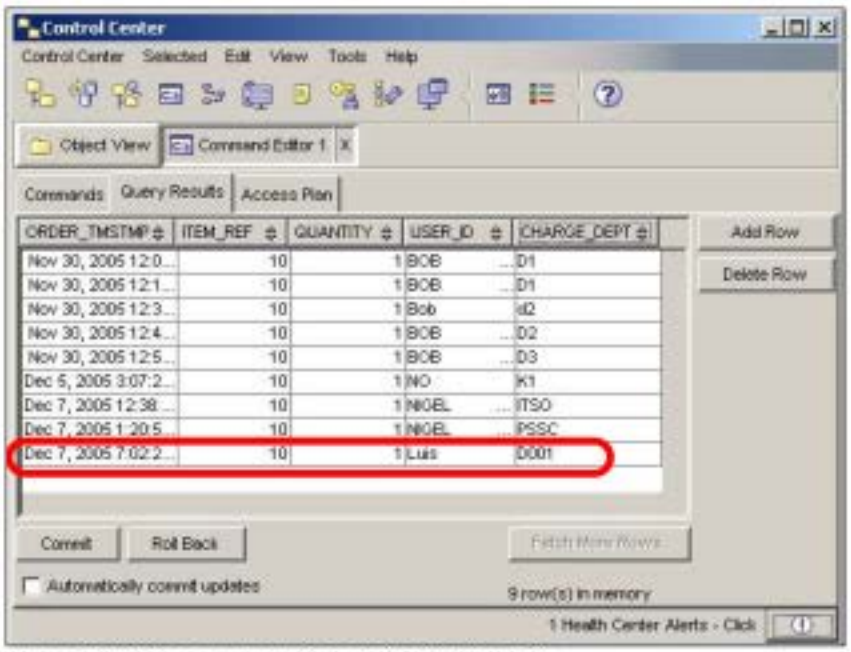


Figure 12-20 The new record in the ITSO .ORDER table

Figure 12-21 shows the registration and protocol service messages that were exchanged between CICS and WebSphere Application Server during our test. Note that WebSphere Application Server uses separate endpoint addresses for its Registration service, Protocol service, and fault messages.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 494/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000494

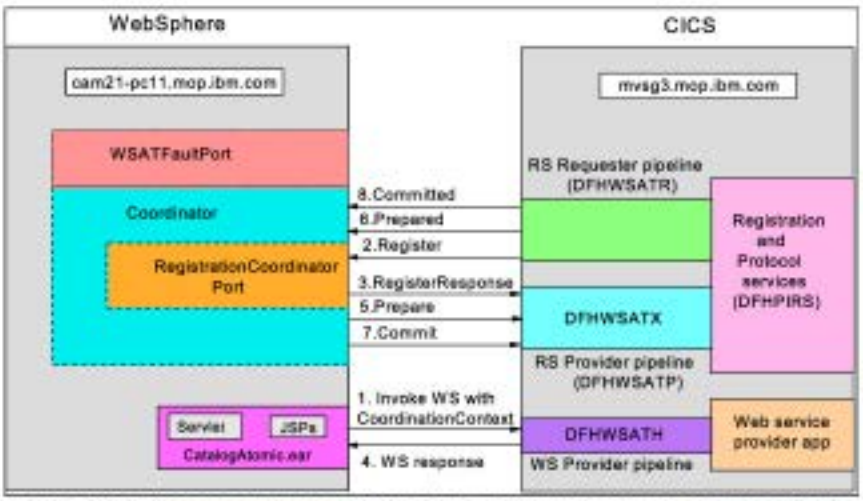


Figure 12-21 Messages exchanged during test of normal atomic transaction termination

We conclude our discussion of this test by showing the complete messages summarized by Figure 12-21. To make each message easier to understand we:

- Format it
- Replace the URL in each xmlns attribute with "..." for each namespace in Example 12-9. For instance, we replaced xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" with xmlns:soap="..." .

Example 12-9 Namespaces used in messages between WebSphere and CICS

```
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wscoor="http://schemas.xmlsoap.org/ws/2004/10/wscoor"
xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension"
xmlns:cicswsat="http://www.ibm.com/xmlns/prod/CICS/pipeline"
```


Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 485/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000495

Invoke Web service with coordination context (message 1)

Example 12-10 shows the SOAP 1.1 message that WebSphere sends to CICS to invoke the placeOrder Web service. The message consists of a SOAP envelope that contains a SOAP header and a SOAP body.

Example 12-10 WebSphere sends to CICS a Web service request with a CoordinationContext header

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
xmlns:xsi="..."
                  xmlns:wscoor="..." xmlns:wsa="...">
   <soapenv:Header>
      <wscoor:CoordinationContext soapen:mustUnderstand="1">
         <wscoor:Expires>never</wscoor:Expires>
         <wscoor:Identifier>
            com.ibm.ws.wstx:0000010810a19b020000000100000006b3e8b0
            ecc989b8eab92e29d91254fd41b0f147b9
         </wscoor:Identifier>
         <wscoor:CoordinationType>
            http://schemas.xmlsoap.org/ws/2004/10/wsat
         </wscoor:CoordinationType>
         <wscoor:RegistrationService xmlns:wscoor="...">
            <wsa:Address xmlns:wsa="...">

http://9.100.199.238:9080/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
            </wsa:Address>
            <wsa:ReferenceProperties xmlns:wsa="...">
               <wsphere-wsat:txID xmlns:wsphere-wsat="...">
                  com.ibm.ws.wstx:0000010810a19b020000000100000006b3e8b0
                  ecc989b8eab92e29d91254fd41b0f147b9
               </wsphere-wsat:txID>
               <wsphere-wsat:instanceID xmlns:wsphere-wsat="...">
                  com.ibm.ws.wstx:0000010810a19b020000000100000006b3e8b0
                  ecc989b8eab92e29d91254fd41b0f147b9
               </wsphere-wsat:instanceID>
            </wsa:ReferenceProperties>
         </wscoor:RegistrationService>
      </wscoor:CoordinationContext>
   </soapenv:Header>
   <soapenv:Body>
      <p635:DFHCICMN xmlns:p635="http://www.DFHCICMN.DFHCICP5.Request.com">
         <p635:ca_request_id>CICSCI</p635:ca_request_id>
         <p635:ca_return_code>0</p635:ca_return_code>
         <p635:ca_response_message></p635:ca_response_message>
         <p635:ca_order_request>
            <p635:ca_userid>Luis</p635:ca_userid>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000496

```
            <p635:ca_charge_dept>DOOI</p635:ca_charge_dept>
            <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
            <p635:ca_quantity_req>1</p635:ca_quantity_req>
            <p635:filler1 xsi:nil="true"/>
         </p635:ca_order_request>
      </p635:DFHOXCMN>
   </soapenv:Body>
</soapenv:Envelope>
```

- SOAP header

  The SOAP header contains only one header, a `CoordinationContext` header. The `CoordinationContext` header has a `mustUnderstand` attribute whose value is `1`. This means that CICS must process the `CoordinationContext` header. If CICS cannot process the `CoordinationContext` header (for example, because the provider pipeline configuration file does not specify the DFHWSATH message handler) then CICS must stop all further processing of the message and generate a SOAP fault whose fault code is `MustUnderstand`.

  The `CoordinationContext` element contains four elements:

  - `Expires`

    CICS treats the content of the `Expires` element as a character string and determines whether the string represents a number.

    - If it does, it is treated as a millisecond value, which is then converted to an integer representing the number of seconds for which the Web service transaction waits for a response to a **`Register`** request or for the Coordinator to send various 2PC messages. If this integer is 0, or greater than 4080, then CICS uses 4080 seconds (the maximum value which CICS allows for the DTIMOUT attribute of the TRANSACTION resource definition).
    - If it does not, then CICS does not set a suspend time and the Web service transaction waits forever.

    Note: We noted that the Expires element was not an unsigned integer but the text `Never`. The default CICS behavior is not to terminate the transaction but to wait indefinitely for WebSphere to commit or roll back the atomic transaction.

  - `Identifier`

    WebSphere creates this identifier as a unique global identifier for each WS-AT transaction as required by the WS-Coordination specification.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000497

CoordinationType

The CoordinationType element specifies the WS-AtomicTransaction coordination type.

- RegistrationService

  The RegistrationService element contains an endpointReference for WebSphere's Registration service. The endpointReference has two elements:

  - Address

    The Address element tells CICS where to send its **Register** request. CICS copies the contents of the Address element to the To message information header when it builds its **Register** request.

  - ReferenceProperties

    WebSphere provides two reference properties: txID and instanceID. The content of the txID element is the same as the content of the Identifier element. The content of the instanceID element is initially the same as that of the txID element, but that will change later.

    CICS adds these reference properties to the SOAP header when it builds its **Register** request. This will allow WebSphere to map the **Register** request to this service request. The detail which WebSphere puts into the txID and instanceID properties is implementation specific and is only ever parsed and understood by WebSphere.

- SOAP body

  The content of the <p635:ca_request_id> element tells CICS which Web service to invoke; 0'ORDR indicates the placeOrder Web service. The contents of the <p635:ca_userid>, <p635:ca_charge_dept>, <p635:ca_item_ref_number>, and <p635:ca_quantity_req> elements provide the placeOrder Web service with the details of the order (see Figure 12-17 on page 463).

Register (message 2)

Example 12-11 shows the SOAP message which contains the **Register** request that CICS sends to WebSphere. The message consists of a SOAP envelope which contains a SOAP header and a SOAP body.

Example 12-11 CICS sends a Register request to WebSphere

```
<soap:Envelope xmlns:wscoor="..." xmlns:wsa="..." xmlns:cicswsat="..."
xmlns:soap="...">
   <soap:Header>
      <wsa:Action>
         http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register
```


Document Id: [illegible] Page 469(of)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000498

```
      </wsa:Action>
      <wsa:MessageID>FIAT MSG A62E73C1 3033431450526270</wsa:MessageID>
      <wsa:ReplyTo>
         <wsa:Address>
            http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
         </wsa:Address>
         <wsa:ReferenceProperties>
            <cicswsat:UOWID>BEC92025168B596B</cicswsat:UOWID>
            <cicswsat:PublicId>
               310FD7E2E2C3C7F34BC1F6D7D6E3F3C3F10FD7E2E2C3C7F34BC1F6D7D6E3F3C3
               F1C3C9F6F2F3C44040AF09202616780000092029167B0C0C4040404040404040
            </cicswsat:PublicId>
         </wsa:ReferenceProperties>
      </wsa:ReplyTo>
      <wsa:To>

http://9.100.199.238:9080/ IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
      </wsa:To>
      <websphere-wsat:txID xmlns:websphere-wsat="...">
         com.ibm.ws.wstx:3030313810a19b020000000100000006d3e8b0
         ecc899b8eab92e29d91254fc4fb0ff47b9
      </websphere-wsat:txID>
      <websphere-wsat:instanceID xmlns:websphere-wsat="...">
         com.ibm.ws.wstx:3030313810a19b020000000100000006d3e8b0
         ecc899b8eab92e29d91254fc4fb0ff47b9
      </websphere-wsat:instanceID>
   </soap:Header>
   <soap:Body>
      <wscoor:Register>
         <wscoor:ProtocolIdentifier>
            http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
         </wscoor:ProtocolIdentifier>
         <wscoor:ParticipantProtocolService>
            <wsa:Address>
               http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
            </wsa:Address>
            <wsa:ReferenceProperties>
               <cicswsat:UOWID>BEC92025168B596B</cicswsat:UOWID>
               <cicswsat:PublicId>
                  310FD7E2E2C3C7F34BC1F6D7D6E3F3C3F10FD7E2E2C3C7F34BC1F6D7D6E3F3C3
                  F1C3C9F6F2F3C44040AF09202616780000092029167B0C0C4040404040404040
               </cicswsat:PublicId>
            </wsa:ReferenceProperties>
         </wscoor:ParticipantProtocolService>
      </wscoor:Register>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000499

```
</soap:Body>
</soap:Envelope>
```

- SOAP header

  The SOAP header contains several message information headers and the two reference properties (txID and instanceID) which WebSphere sent in its Web service request. These reference properties allow WebSphere to match the **Register** request to its initial request for the placeOrder service. The message information headers are as follows.

  - To

    The To header shows that the message is being sent to the Registration Coordinator Port running in a WebSphere Application Server V6.0 region which is monitoring port 9080 on a system whose IP address is 9.100.199.238. CICS copied this from the Address element of the endpointReference for the RegistrationService in WebSphere's service request.

  - Action

    The Action header indicates that CICS wishes to register with the Registration Coordinator Port.

  - MessageID

    The ID of the message uniquely identifies the message in space and time:

    - PIAT is part of the name of module DFHPIAT. DFHPIAT generates CoordinationContext elements for CICS and interprets CoordinationContext elements received from other systems.
    - A6P0T3C1 is the VTAM APPLID of the CICS region which is sending the message.
    - [illegible] is the abstime value returned by an EXEC CICS INQUIRE TIME issued in our CICS region.

  - ReplyTo

    The Address element of the ReplyTo header shows that the reply to this message should be sent to the Registration service running in a CICS region which is monitoring port 15301 on a z/OS system whose IP address is MVSG3.mop.ibm.com. This Registration service has two reference properties:

    - UOWID
    - PublicID

    When WebSphere sends the **RegisterResponse** to CICS, it will add each of these reference properties to the response as a SOAP header. CICS

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000500

will use the PublicID to find the region and the UOW in that region where the Web service provider is waiting for the response; then CICS will route it there for processing. The detail which CICS puts into the UOWID and PublicID properties is there for CICS to understand; other products do not use it.

- SOAP body

  The SOAP body contains the **Register** request, which contains two elements:

  ProtocolIdentifier

  CICS uses this element to register for the durable two-phase commit protocol

  - ParticipantProtocolService

    Note that the address of CICS's Protocol service is the same as the address of its Registration service.

RegisterResponse (message 3)

Example 12-12 shows the **RegisterResponse** that WebSphere sends to CICS.

Example 12-12 WebSphere sends a RegisterResponse to CICS

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
xmlns:xsi="..."
                  xmlns:wsa="...">
   <soapenv:Header>
      <wsa:MessageID>uuid:1CA19035-0108-4000-E000-094403D46363</wsa:MessageID>
      <wsa:To>http://MVSG3.mop.ibm.com:15301/c:cswsal/RegistrationService</wsa:To>
      <wsa:Action>
         http://schemas.xmlsoap.org/ws/2004/10/wscoor/RegisterResponse
      </wsa:Action>
      <wsa:FaultTo xmlns:wsa="...">
         <wsa:Address>
            http://9.103.199.230:9080/ IBMSYSAPP/wsalfault/services/WSATFault?wsdl
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere-wsat:txID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:00000106101a19b020000000100000068Je6b0
               ec:899b8ea92e29d91254fc4fb0ff47a9
            </websphere-wsat:txID>
            <websphere-wsat:instanceID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:00000106101a19b020000000100000068Je6b0
               ec:899b8ea92e29d91254fc4fb0ff47a9
            </websphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:FaultTo>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000501

```
      <wsa:From xmlns:wsa="...">
         <wsa:Address>

http://9.100.199.238:9080/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere wsat:txID xmlns:websphere wsat="...">
               com.ibm.ws.wsat:0000010810a19b320D3D3D310D0D0D0E83e8b0
               ecc889b8eab92e29d91254fc4fb01f47b9
            </websphere-wsat:txID>
            <wshsphere-wsat:instanceID xmlns:wshsphere-wsat="...">
               com.ibm.ws.wsat:0000010810a19b320D0D0D0D10D0D0D0E83e8b0
               ecc889b8eab92e29d91254fc4fb01f47b9
            </websphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:From>
      <wsa:RelatesTo>F1AT MSG A6P3T3C1 3033431450526270</wsa:RelatesTo>
      <cicswsat:UOWID xmlns:cicswsat="..."><E3920Z5168B5956</cicswsat:UOWID>
      <cicswsat:PublicId xmlns:cicswsat="...">
         31CFD7E2E2C3C7F348C1F607D6E3F3C3F1C0F07E2E2C3C7F34BC1F6070E3F3C3
         F1C3C9F6F2F3C44040F0970261678000009702616780000404040404040404040
      </cicswsat:PublicId>
   </soapenv:Header>
   <soapenv:Body>
      <RegisterResponse xmlns="http://schemas.xmlsoap.org/ws/2004/10/wscoor">
         <wscoor:CoordinatorProtocol$ervice xmlns:wscoor="...">
            <wsa:Address xmlns:wsa="...">
               http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator
            </wsa:Address>
            <wsa:Reference Properties xmlns:wsa="...">
               <websphere-wsat:txID xmlns:websphere-wsat="...">
                  com.ibm.ws.wsta:0000010810a19b320D0D0D0D10D0D0D0E83e8b0
                  ecc889b8eab92e29d91254fd41b3f47b9
               </websphere-wsat:txID>
               <websphere-wsat-instanceID xmlns:websphere-wsat="...">
                  http://W5G3.mop.ibm.com:16301/cicswsat/RegistrationService
               </websphere wsat:instanceID>
            </wsa:ReferenceProperties>
         </wscoor:CoordinatorProtocol Service>
      </RegisterResponse>
   </soapenv:Body>
</soapenv:Envelope>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000502

- SOAP header

  The SOAP header contains several message information headers and the two reference properties (UOW[]) and Public[]) which CICS sent in its **Register** request. If we were running a CICSPlex, CICS would use these reference properties to route the response to the region and the UOW in that region where the Web service provider is waiting for it. The message information headers are as follows:

  - MessageID

    To

    WebSphere is sending this message to CICS's RegistrationService. It obtained this address from the ReplyTo header in the **Register** request.

  - Action

    This message is a **RegisterResponse**.

    FaultTo

    WebSphere wants CICS to send any SOAP faults that it has to generate to WebSphere's WSATFaultPort.

  - From

    This message is coming from WebSphere's RegistrationCoordinatorPort.

    RelatesTo

    This message relates to the **Register** request which CICS sent.

- SOAP body

  The SOAP body contains the RegisterResponse element, which in turn contains only a CoordinatorProtocolService element. The address of WebSphere's Protocol Service is http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator. Note that its two reference properties, txID and instanceID, no longer have the same contents, instanceID now contains the address of CICS's Registration service.

Web service response (message 4)

Since CICS has registered its interest in the atomic transaction and WebSphere has responded to that, CICS can now run the placeOrder service. When the placeOrder service has completed its work, CICS sends a response to WebSphere as shown in Example 12-13. The essence of the response is the message ORDER SUCCESSFULLY PLACED.

Example 12-13 CICS sends the Web service response to WebSphere

```
<SOAP-ENV:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000503

```
                xmlns:xsi="..."     xmlns:wscoor="..."  xmlns:wsa="..."
                xmlns:SOAP-ENV="...">
   <SOAP-ENV:Body>
      <DFHOXCMNResponse xmlns="http://www.DFHOXCMN.DFHOXCP5.Response.com">
         <ca_request_id>01ORDR</ca_request_id>
         <ca_return_code>0</ca_return_code>
         <ca_response_message>ORDER SUCESSFULLY PLACED</ca_response_message>
         <ca_order_request>
            <ca_userid>Luis     </ca_userid>
            <ca_charge_dept>0001    </ca_charge_dept>
            <ca_item_ref_number>10</ca_item_ref_number>
            <ca_quantity_req>1</ca_quantity_req>
            <filler>
            ...
            </filler>
         </ca_order_request>
      </DFHOXCMNResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Prepare (message 5)

When the handlePlaceOrder method of the CatalogController servlet issues the **userTransaction.commit** command, WebSphere sends a **Prepare** command to CICS as shown in Example 12-14 in the Action message information header and in the <p320:Prepare.../> element of the SOAP Body.

Example 12-14 WebSphere sends a Prepare notification to CICS

```
<soapenv:Envelope   xmlns:soapenv="..."   xmlns:soapenc="..."   xmlns:xsd="..."
                    xmlns:xsi="..."       xmlns:wsa="...">
   <soapenv:Header>
      <wsa:MessageID>uuid:1DA19D3B 3D36 4DD1 E000 0944D9D4B359</wsa:MessageID>
      <wsa:To>http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationServices</wsa:To>
      <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepare</wsa:Action>
      <wsa:FaultTo xmlns:wsa="...">
         <wsa:Address>
            http://9.100.199.230:3080/ IBMSYSAPP/wsatfault/services/WSATFault?wsdl
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere-wsat:txID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:0000010810a:9b070000000100000000683e6b0
               ec:c809b0ea:92e29:91254fc4:0:0:1:47b9
            </websphere-wsat:txID>
            <websphere-wsat:instanceID xmlns:websphere-wsat="...">
               http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000504

```
            </wsbsphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:FaultTo>
      <wsa:ReplyTo xmlns:wsa="...">
         <wsa:Address>
            http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere-wsat:txID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:30303130610a19b020000030010000000083e8b0
               ecc899h8eah92c29d91254fc4fh0ff47a9
            </websphere-wsat:txID>
            <websphere-wsat:instanceID xmlns:websphere-wsat="...">
               http://MVS33.nap.ibm.com:1530L/cicswsat/RegistrationService
            </websphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:ReplyTo>
      <cicswsat:UOWID xmlns:cicswsat="...">BE39202516dB5956</cicswsat:UOWID>
      <cicswsat:PublicId xmlns:cicswsat="...">
         31CFD7E2E2C3C7F340C1F607D6E3F3C3F1C0D7E2E2C3C7F34BC1FC070CE3F3C3
         F1C3C9F6F2F3C44040AF39702516760000097025167B0000404040404040404040
      </cicswsat:PublicId>
   </soapenv:Header>
   <soapenv:Body>
      <p320:Prepare xsi:nil="true"
xmlns:p320="http://schemas.xmlsoap.org/ws/2004/10/wsat"/>
   </soapenv:Body>
</soapenv:Envelope>
```

Prepared (message 6)

CICS responds with **Prepared** as shown in Example 12-15 in the Action message information header and in the <wsat:Prepared.../> element of the SOAP Body. Since **Prepared** is a terminating message, it does not contain a ReplyTo message information header.

Example 12-15 CICS sends Prepared notification to WebSphere

```
<soap:Envelope xmlns:xscoor="..." xmlns:wsa="..." xmlns:cicswsat="..."
xmlns:soap="...">
   <soap:Header>
      <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepared</wsa:Action>
      <wsa:MessageID>FEAT MSG A6PST3C1 30334314505327DC</wsa:MessageID>
      <wsa:To>http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator</wsa:To>
      <websphere-wsat:txID xmlns:websphere-wsat="...">
         com.ibm.ws.wstx:30303130610a19b020000030010000000083e8b0
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000505

```
            ecc889b8eab92e29d91254fc4fb0ff47b9
        </websphere-wsat:txID>
        <websphere-wsat:instanceID xmlns:websphere-wsat="...">
            http://MVS03.mop.ibm.com:15301/cicswsat/RegistrationService
        </websphere-wsat:instanceID>
    </soap:Header>
    <soap:Body>
        <wsat:Prepared xmlns:wsat="..."/>
    </soap:Body>
</soap:Envelope>
```

Commit (message 7)

Since CICS has voted **yes** in response to the **Prepare** command, and since no other system has registered an interest in this transaction, WebSphere sends a **Commit** command to CICS. See the Action header and the <p320:Commit.../> element of the SOAP Body in Example 12-16

Example 12-16 WebSphere sends Commit notification to CICS

```
<soapenv:Envelope xmlns:soapenv="..."  xmlns:soapenc="..." xmlns:xsd="..."
xmlns:xsi="..."
                  xmlns:wsa="...">
   <soapenv:Header>
      <wsa:MessageID>uuid:10A19F67-1C38-4000-E000-09F409D4B353</wsa:MessageID>
      <wsa:To>http://MVS03.mop.ibm.com:15301/cicswsat/RegistrationService</wsa:To>
      <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Commit</wsa:Action>

      <wsa:FaultTo xmlns:wsa="...">
         <wsa:Address>
            http://9.100.199.238:9080/_IBMSYS4PP/wsatfault/services/WSATFaultPort
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere-wsat:txID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:00000C0810a19b020000000100000000683eBb0
               ecc889b8eab92e29d91254fc4fb0ff47b9
            </websphere-wsat:txID>
            <websphere-wsat:instanceID xmlns:websphere-wsat="...">
               http://MVS03.mop.ibm.com:15301/cicswsat/RegistrationService
            </websphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:FaultTo>
      <wsa:ReplyTo xmlns:wsa="...">
         <wsa:Address>
            http://9.100.199.238:9080/_IBMSYS4PP/wsat/services/Coordinator
         </wsa:Address>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000506

```
         <wsa:ReferenceProperties xmlns:wsa="...">
            <webSphere wsat:txID xmlns:webSphere wsat="...">
               com.ibm.ws.wstx:00000130810a19b320000000010000000E83e8b0
               ecc889b8eab92e29d91254fc4fb0ff47b9
            </webSphere-wsat:txID>
            <webSphere-wsat:instanceID xmlns:webSphere-wsat="...">
               http://MVS53.mop.ibm.com:1530I/cicswsat/RegistrationService
            </webSphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:ReplyTo>
      <cicswsat:UOWID xmlns:cicswsat="...">9E092026168B566B</cicswsat:UOWID>
      <cicswsat:PublicId xmlns:cicswsat="...">
         31CFD7E2E2C3C7F34BC1F60706E3F3C3F1CF37E2E2C3C7F34BC1FED7DEE3F3C3
         F1C3C9E6E2F3C4404D8E3D9202516760D0D09202516760D0D0404040404040404040
      </cicswsat:PublicId>
   </soapenv:Header>
   <soapenv:Body>
      <p320:Commit xsi:nil="true"
xmlns:p320="http://schemas.xmlsoap.org/ws/2004/10/wsat"/>
   </soapenv:Body>
</soapenv:Envelope>
```

Committed (message 8)

After committing the update to the EXMPCAT VSAM file, CICS sends a **Committed** notification to WebSphere. See the Action message information header and the <wsat:Committed .../> element of the SOAP Body in Example 12-17. Since **Committed** is a terminating message, it does not contain a ReplyTo message information header.

Example 12-17 CICS sends Committed notification to WebSphere

```
<soap:Envelope   xmlns:wsaw="..."   xmlns:wsa="..."   xmlns:cicswsat="..."
                 xmlns:soap="...">
   <soap:Header>
      <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Committed</wsa:Action>
      <wsa:MessageID>PT4T-H5G-A6P3T3C1-3033431460533240</wsa:MessageID>
      <wsa:To>tcp://9.100.199.230:9080/ IBMSYSAPP/wsat/services/Coordinator</wsa:To>
      <webSphere-wsat:txID xmlns:webSphere-wsat="...">
         com.ibm.ws.wstx:00000130810a19b320000000010000000E83e8b0
         ecc889b8eab92e29d91254fc4fb0ff47b9
      </webSphere-wsat:txID>
      <webSphere wsat:instanceID xmlns:webSphere wsat="...">
         http://MVS53.mop.ibm.com:1530I/cicswsat/RegistrationService
      </webSphere-wsat:instanceID>
   </soap:Header>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000507

```
   <soap:Body>
      <wsat:Committed xmlns:wsat="..."/>
   </soap:Body>
</soap:Envelope>
```

Let's think back to Waldo and his money transfer transaction (see "Beginner's guide to atomic transactions" on page 382). What Waldo requires is that both databases cooperate to ensure that they are always in a known and consistent state.

The set of eight Web service, registration, and protocol service messages we have described here are the flows that guarantee database consistency in our scenario, a scenario in which the databases are accessed by middleware (WebSphere Application Server on a Windows platform and CICS on z/OS), and communication between the middleware products is based on open Web services standards. The same basic flows could be used between any middleware products which support the Web services, Web Services-Coordination (WS-C) and Web Services-Atomic Transaction (WS-AT) specifications.

Simple scenario: abnormal transaction termination

We return to the Place Order window to place a new order. This time we used ROLLBACK as the User Name. This tells the servlet to throw an exception after inserting the order in the DB2 table and placing the order with CICS.

Example 12-18 CatalogController servlet: Throwing the RemoteException

```
if(order.getUserId().equalsIgnoreCase("ROLLBACK"))
{
   System.out.println("CatalogController.handlePlaceOrder() - Simulating the
RemoteException");
   throw new RemoteException("Throwing the RemoteException");
}
```

On the Order Entry page we entered the User Name ROLLBACK and clicked **SUBMIT** to place a new order (Figure 12-22). Note that, at this time, we still had 75 items in stock (see Figure 12-19 on page 464).

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000508



Figure 12-22 Catalog Application - Place order for ROLLBACK

Figure 12-23 shows the expected error response.

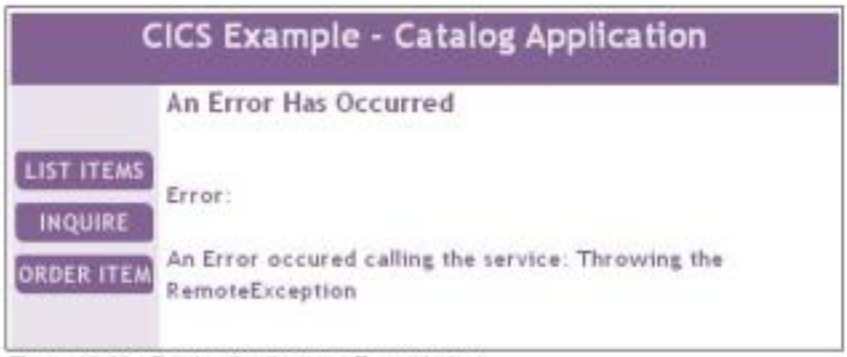


Figure 12-23 Catalog Application - Error window

When the Exception is thrown, the transaction is rolled back; the order is deleted from the table and the CICS transaction is rolled back too.

We again check in the WebSphere server log to see what happened (Example 12-19).

Example 12-19 WebSphere server log with ROLLBACK Place Order

```
CatalogContoller:doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
```


Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineHistory.net
Page 509/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000509

```
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - simulating the RemoteException
CatalogController.handlePlaceOrder() - rollingback the transaction
```

We can see in the log that the order was inserted in the database and a good response was returned from calling the CICS Web service. Then we simulated the RemoteException and the rollback of the whole transaction took place.

When we inquired on the stock level, we noted that there were still 75 items in stock.

Figure 12-24 shows the registration and protocol service messages that CICS and WebSphere Application Server exchanged during our test.

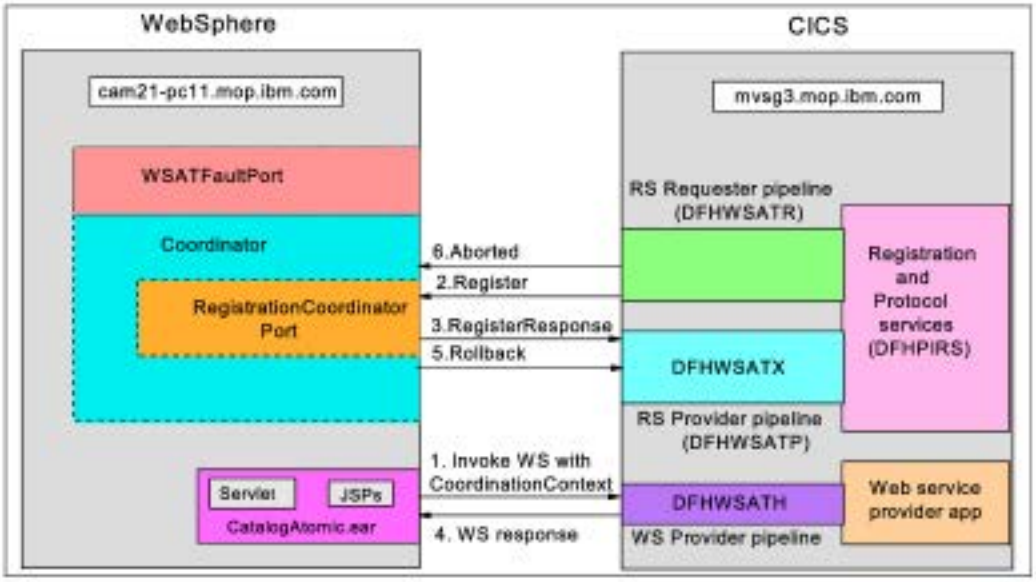


Figure 12-24 Messages exchanged when both sides back out

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 510/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000510

12.3 The daisy chain atomic transaction scenario

We now extend the simple atomic transaction scenario to a *daisy chain* scenario by making the following modifications:

- Changing the configuration of the CICS sample Catalog application such that it makes an outbound Web service call
- Changing the ExampleAppDispatchOrder.ear file provided with CICS TS V3.1

  ExampleAppDispatchOrder.ear is an enterprise archive provided with the Catalog application that can be deployed in WebSphere Application Server and used as an order dispatch endpoint. See "Catalog manager example application" on page 67 for more information on the sample application.

  We modified the ExampleAppDispatchOrder.ear file to insert records into a DB2 table called ITSO.DISPATCH. The ITSO.DISPATCH table logs all of the orders dispatched through the dispatchOrder Web service. We called the modified program DispatchOrderAtomic.ear.

By making these changes, we have a global transaction that updates three resources:

- A DB2 table in the Windows environment
- A VSAM file in the z/OS environment
- Another DB2 table in a Windows environment

We have a daisy chain from WebSphere to CICS to WebSphere. CICS acts as both a coordinator and a participant. Figure 12-25 shows the sequence of events for the whole transaction.

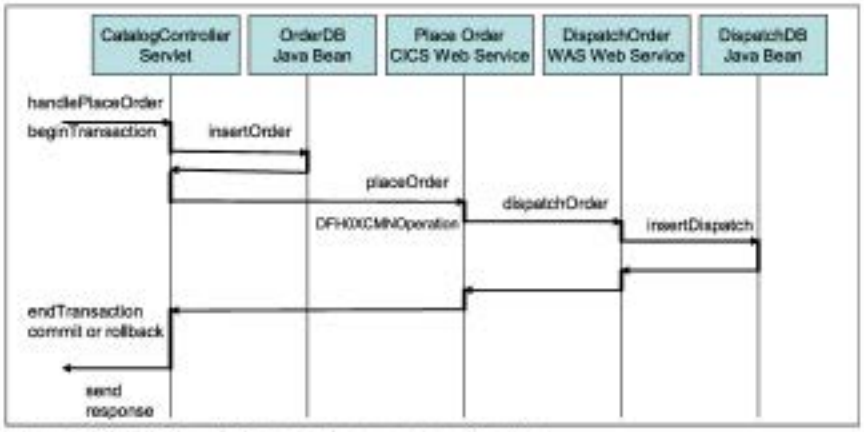


Figure 12-25 Daisy chain scenario sequence of events

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 511/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000511

Figure 12-26 shows a more global view of the sequence of events

- The user uses his Web browser to invoke the AtomicClient CatalogController servlet, which runs in WebSphere Application Server
- The AtomicClient updates the ITSO.ORDER table in DB2.
- WebSphere Application Server sends the placeOrder SOAP message containing the order to CICS.
- CICS uses the Web services provider PIPELINE definition PIPE1 to process the SOAP message. PIPE1 contains our SNIFFER program and the CICS-supplied SOAP 1.1 message handler DFHPISN1.
- DFHPISN1 links to the CICS-supplied header processing program DFHWSATH.
- CICS converts the SOAP message to a COMMAREA for the sample catalog manager program, DFH0XCMN.
- DFH0XCMN passes the data in the COMMAREA to the sample catalog program DFH0XVDS, which updates the recoverable VSAM file.
- DFH0XCMN passes the data in the COMMAREA to DFH0XWOD
- DFH0XWOD invokes the dispatchOrder Web service, which points to the PIPELINE PIPE2
- PIPE2 contains DFHPISN1, which:
  - Converts the COMMAREA to a SOAP message body
  - Calls DFHWSATH to create a CoordinationContext SOAP header
  - Sends the dispatchOrder SOAP message to WebSphere
- DispatchOrderAtomic updates the ITSO.DISPATCH table in DB2.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000512

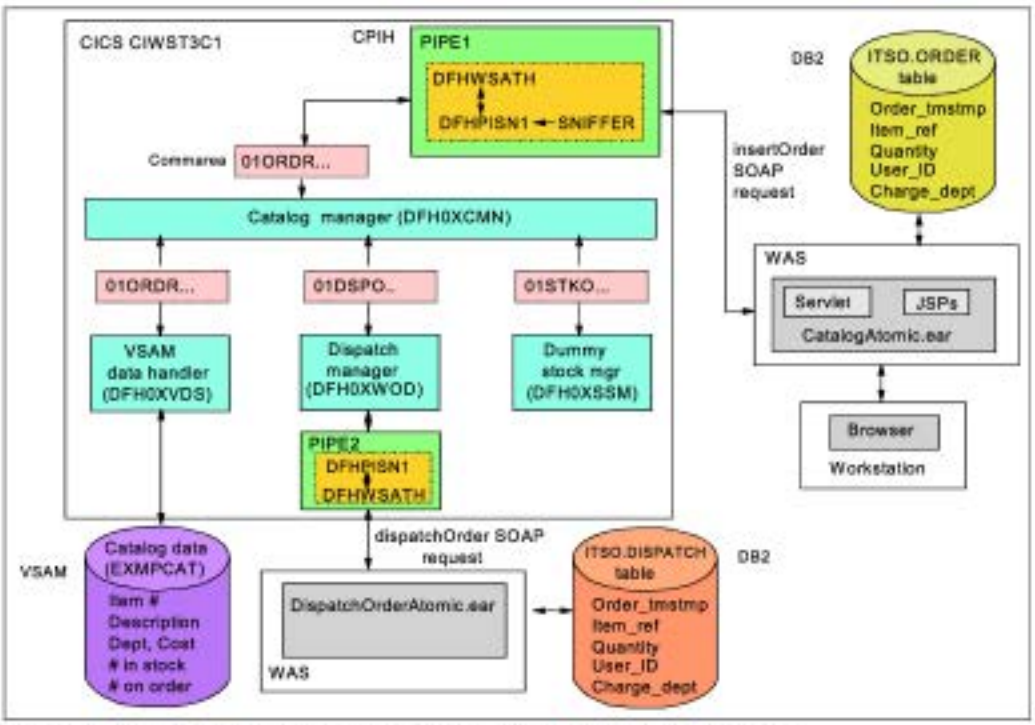


Figure 12-26 Daisy chain: CICS as a service provider and as a service requester

In the following sections we describe how we set up CICS for the daisy chain scenario, and how we created the DispatchOrderAtomic application and the ITSO.DISPATCH table. We then show the results of testing this scenario.

12.3.1 Setting up CICS for the daisy chain scenario

To set up CICS for this scenario we performed the following steps in addition to those shown in 12.2.1, "Setting up CICS for the simple scenario" on page 441:

- We edited PIPE2's configuration file /CIWS/T3C1/config/ITSO_7206_wsat_soap11request.xml

  It now contains the XML shown in Example 12-20.

Example 12-20 PIPE2:/CIWS/T3C1/config/ITSO_7206_wsat_soap11request.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000513

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
    requester.xsd ">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler>
        <header_programs>
          <program_name>D-HWSA-H</program_name>
          <namespace>
             http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>true</mandatory>
        </header_programs>
      </cics_soap_1.1_handler>
    </service_handler_list>
  </service>
  <default_transport_handler_list>
    <handler>
      <program>SHIFTCA</program>
      <handler_parameter_list/>
    </handler>
  </default_transport_handler_list>
  <service_parameter_list>
    <registration_service_epr_address>
       http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
    </registration_service_epr_address>
  </service_parameter_list>
</requester_pipeline>
```

- We modified the configuration file for the sample Catalog application using the ECHO transaction at a 3270 terminal connected to our CICS region (see Figure 3-17 on page 98). In the 3270 screen titled Configure CICS Example Catalog Application:
  - We set the value of the Outbound WebService? field on this screen to **Yes**; this causes the Catalog Manager program (DFH0XCMN) to invoke the Dispatch Manager program (DFH0XWOD) rather than the Dummy Dispatch Manager program (DFH0XSOD) that we used in the simple scenario. While DFH0XSOD simply sets the return code in the COMMAREA to 0 and returns to its caller, DFH0XWOD issues an **EXEC CICS INVOKE WEBSERVICE('dispatchOrder') URI(outboundWebServiceURI)** command to make an outbound Web service call to an order dispatcher
  - We set the value of the Outbound WebService URI field to the location of the Web service that implements the order dispatcher function. We ran the dispatchOrder service on WebSphere Application Server for Windows.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000514

12.3.2 Creating DispatchOrderAtomic and the ITSO.DISPATCH table

To create the DispatchOrderAtomic application and the ITSO.DISPATCH table, we did the following:

- Created the DispatchDB JavaBean that inserts the dispatch order into the DB2 table.
- Changed the dispatchOrder method in the DispatchOrderSoapBindingImpl class in order to call the DispatchDB JavaBean.
  DispatchOrderSoapBindingImpl is the bean that serves the Web service and dispatchOrder is the method that is called when the Web service is requested.
- Changed the deployment descriptor of the DispatchOrderAtomic application so that it is executed as part of a Web Services Atomic Transaction.
- Created the ITSO.DISPATCH table.

We explain each step in detail in the following sections.

Create the DispatchDB JavaBean

In the same way that we previously created the OrderDB JavaBean, we created the DispatchDB JavaBean. This bean has a method that inserts the dispatch order into the ITSO.DISPATCH table. The inp1 parameter is an OrderBean that has all the data needed for the insert. We do not show the DispatchDB JavaBean code here because it is very similar to the OrderDB code shown in Example 12-4 on page 446.

Change the dispatchOrder method

When the dispatchOrder Web service is called, the dispatchOrder method in the DispatchOrderSoapBindingImpl Java class is called. It is responsible to perform the service and send a response. We changed this method to create an OrderBean and to perform the insert by calling the insertDispatch method in the DispatchDB JavaBean.

The dispatchOrder method is shown in Example 12-21.

Example 12-21 The dispatchOrder method

```
public com.Response.dispatchOrder.exampleApp.www.DispatchOrderResponse
   dispatchOrder(com.Request.dispatchOrder.exampleApp.www.DispatchOrderRequest
requestPart) throws java.rmi.RemoteException {
      System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder(): "+
            " ItemRef="+requestPart.getItemReferenceNumber()+
            " Quantity="+requestPart.getQuantityRequired()+
            " CustomerName="+requestPart.getCustomerId()+
            " Dept="+requestPart.getChargeDepartment());
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000515

```
        // Creating the order
        System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder() -
creating the order");
        OrderBean order = new OrderBean();
        order.setOrderTimestamp(new
Timestamp(Calendar.getInstance().getTime().getTime()));
        order.setItemRef(requestPart.getItemReferenceNumber());
        order.setQuantity(requestPart.getQuantityRequired());
        order.setUserId(requestPart.getCustomerId());
        order.setChargeDept(requestPart.getChargeDepartment());

        // Insert to the order in the database
        System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder() -
inserting the order in the database");
        DispatchDB dispatchDB = new DispatchDB();
        dispatchDB.insertDispatch(order);
        System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder() - after
the insert!!!");

        com.Response.dispatchOrder.example5pp.www.DispatchOrderResponse
        response = new
com.Response.dispatchOrder.example5pp.www.DispatchOrderResponse();
        response.setConfirmation("Order in Dispatch");
          return response;
    }
```

Change the deployment descriptor

In order to specify that the dispatchOrder service should participate in the global transaction, we needed to modify the Web Application deployment descriptor.

To activate WS-AT support:

- We imported the application archive DispatchOrderAtomic.ear into RAD. We then expanded the **Dynamic Web Project** (DispatchOrderAtomicWeb) in the Project Explorer and opened (double-clicked) the **Deployment Descriptor** (Figure 12-27)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000516

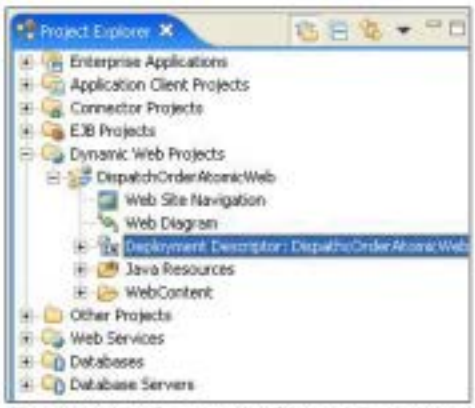


Figure 12-27 Opening the DispatchOrderAtomicWeb Deployment Descriptor

- In the Deployment Descriptor we clicked the **Servlets** tab and then we selected the **com_dispatchOrder_exampleApp_www_DispatchOrderSoapBindingImpl** servlet. Then we scrolled down to Global Transaction and selected **Execute using Web Services Atomic Transaction on incoming request** (Figure 12-28).

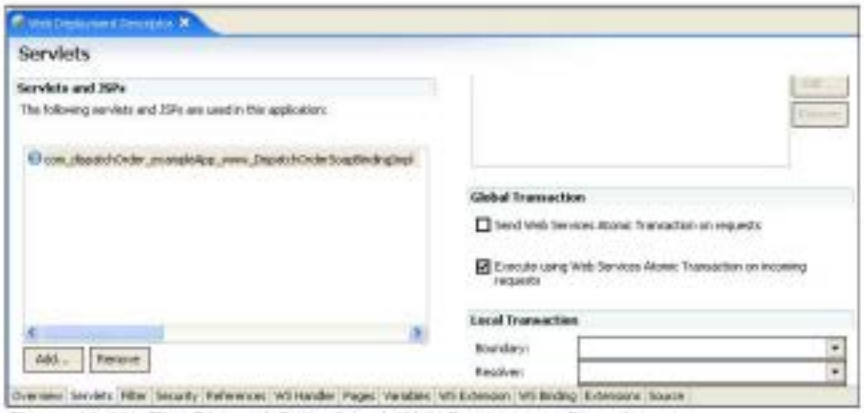


Figure 12-28 The DispatchOrderAtomicWeb Deployment Descriptor

- We saved and closed the file.

Now when an incoming request contains a coordination context, the dispatchOrder Web service will join the global transaction.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 517/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000517

Create the ITSO.DISPATCH table

We created the ITSO.DISPATCH table to record all the dispatch orders that come to the dispatchOrder Web service. Example 12-22 shows the script we used to create the table.

Example 12-22 Creation of ITSO.DISPATCH table

```
   DB2 CLP

CONNECT RESET;

CONNECT TO ITSODB;

-- Table definitions for Dispatch
CREATE TABLE ITSO.DISPATCH
  (DISPATCH_TMSTMP TIMESTAMP  NOT NULL ,
   ITEM_REF INTEGER  NOT NULL ,
   QUANTITY INTEGER  NOT NULL ,
   USER_ID CHARACTER (20)  NOT NULL ,
   CHARGE_DEPT CHARACTER (20)  NOT NULL  ,
   CONSTRAINT DISPATCHKEY PRIMARY KEY (DISPATCH_TMSTMP)  ) ;
```

The ITSO.DISPATCH table has a column for every field in the Catalog Place Order JSP. Also, there is a TimeStamp column that is used as a unique key for the table.

12.3.3 Testing the daisy chain scenario

We performed two tests of the daisy chain scenario:

- Normal transaction termination
- Abnormal transaction termination (see 'Daisy chain scenario: abnormal transaction termination' on page 495)

Daisy chain scenario: normal transaction termination

In this section, we explain how we ran the AtomicClient application and then show the registration and protocol service messages that are exchanged between WebSphere Application Server and CICS during the normal termination of the atomic transaction.

To run the scenario we did the following:

- Opened the welcome page of the Catalog application.

  http://cam21-pc11:9080/CatalogAtomicWeb/welcome.jsp

- Clicked **INQUIRE** to perform an inquireSingle operation.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000518

- In the **Inquire Single** window we used the Item Reference Number default value of 0010 and clicked **SUBMIT**. The Web service request was sent to CICS and we were presented with the results of the inquiry as shown in Figure 12-29.



Figure 12-29 Catalog Application - Inquire single before

- We noted that the number of items in stock was 31. This value is taken from the CICS VSAM file.
- We clicked **SUBMIT** to go to the Enter Order Details window shown in Figure 12-30.



Figure 12-30 Catalog Application - Enter order details

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 519/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000519

- In the Enter Order Details window we provided a User Name and a Department Name and clicked **SUBMIT**.
- After the CICS Web service processed the order and called the dispatchOrder WebSphere Web service, we got the response telling us that the order was successfully placed (Figure 12-31).



Figure 12-31 Catalog Application - Order successfully placed

- In the WebSphere server log we see trace entries generated by the CatalogController servlet, which show that the transaction was successfully committed (Example 12-23).

Example 12-23 WebSphere server log with successful Place Order for CatalogController

```
CatalogContoller:doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - commit the transaction
CatalogController.handlePlaceOrder() - after commit
```

- We also can see the log for the WebSphere Application Server on which the DispatchOrderAtomic application is installed (Example 12-24).

Example 12-24 WebSphere server log with successful Place Order for DispatchOrder

```
DispatchOrderSoapBindingImpl.dispatchOrder(): ItemRef=10 Quantity=1
CustomerName=Luis     Dept=Itso
DispatchOrderSoapBindingImpl.dispatchOrder() - creating the order
DispatchOrderSoapBindingImpl.dispatchOrder() - inserting the order in the
database
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 520/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000520

```
DispatchDB.insertDispatch() - inserted the order in the database!!!!
DispatchOrderSoapBindingImpl.dispatchOrder() - after the insert!!!
```

- Next we checked the same item number through the inquireService service and verified that the stock level decreased by one item (Figure 12-32).



Figure 12-32 Catalog Application - Inquire single after

- We opened the DB2 Control Center and used the following SQL command on the Windows machine which hosted the ITSO.ORDER table:

  ```
  SELECT * FROM ITSO.ORDER
  ```

  Figure 12-33 shows the new record in the table.

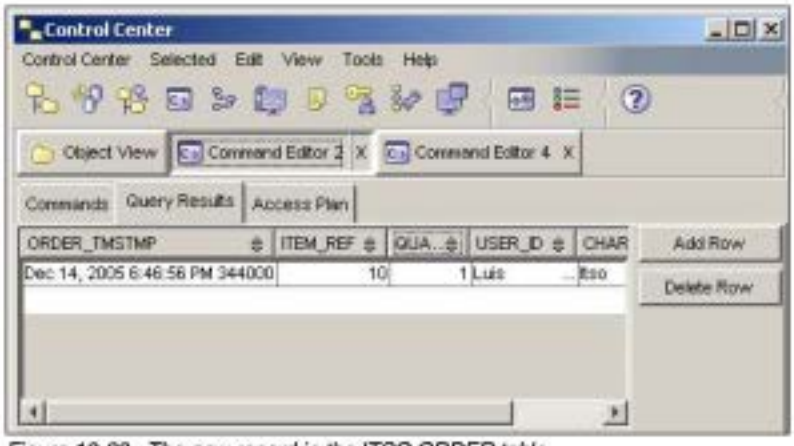


Figure 12-33 The new record in the ITSO.ORDER table

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 521/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000521

- We also used the following SQL command on the Windows machine which hosted the ITSO.DISPATCH table:

  ```
  SELECT * FROM ITSO.DISPATCH
  ```

  Figure 12-34 shows the new record in the table.

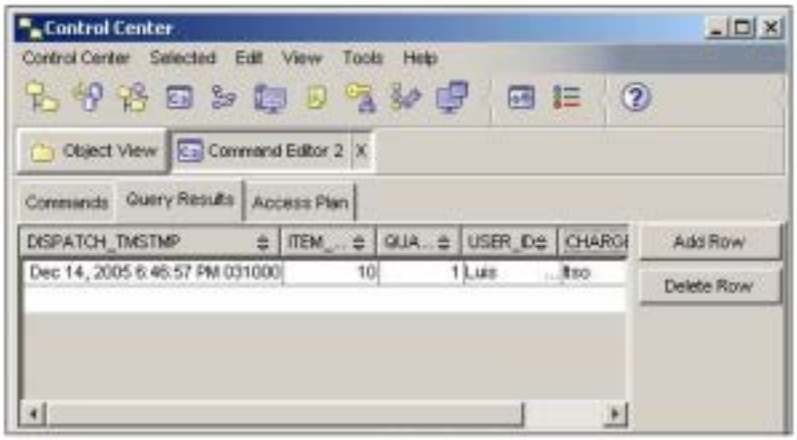


Figure 12-34 The new record in the ITSO.DISPATCH table

Figure 12-35 shows the registration and protocol service messages that are exchanged between CICS and the two WebSphere Application Servers during our test.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 522/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000522

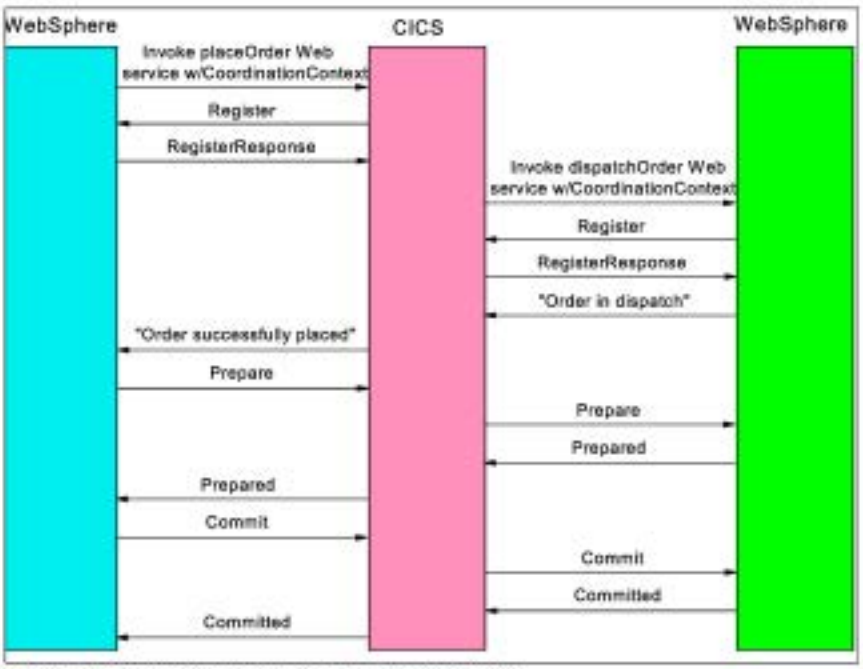


Figure 12-35 Successful daisy chain - Message flow

Example 12-25 shows the dispatchOrder request that CICS sends to WebSphere. Recall that when CICS is a *participant* in a WS-AT transaction, it uses two reference properties: `UOWID` and `PublicID`. We see in Example 12-25 that when CICS is the *coordinator*, it uses three reference properties: `UOWID`, `Token`, and `Netname`.

Example 12-25 CICS sends dispatchOrder request to WebSphere

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="..." xmlns:wscoor="..." xmlns:wsa="..."
xmlns:cicswsat="..."                   xmlns:soap="...">
   <SOAP-ENV:Header>
      <wscoor:CoordinationContext>
         <wscoor:Identifier>
            PIAT-CCON-A6POT3C1-003343578075119C
         </wscoor:Identifier>
         <wscoor:CoordinationType>
            http://schemas.xmlsoap.org/ws/2004/10/wsat
         </wscoor:CoordinationType>
         <wscoor:RegistrationService>
```


Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000523

```
            <wsa:Address>
               http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
            </wsa:Address>
            <wsa:ReferenceProperties>
               <cicswsat:Netname>A6P0T3C1</cicswsat:Netname>
               <cicswsat:Token>F0F0F0F0</cicswsat:Token>
               <cicswsat:UOWID>BE0F69B097751067</cicswsat:UOWID>
            </wsa:ReferenceProperties>
         </wscoor:RegistrationService>
      </wscoor:CoordinationContext>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <dispatchOrderRequest xmlns="http://www.exampleApp.dispatchOrder.Request.com">
         <itemReferenceNumber>10</itemReferenceNumber>
         <quantityRequired>1</quantityRequired>
         <customerId>Luis     </customerId>
         <chargeDepartment>Itso     </chargeDepartment>
      </dispatchOrderRequest>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Daisy chain scenario: abnormal transaction termination

Finally, we tested the daisy chain scenario, but this time with a RemoteException thrown at the end of the execution. We returned to the Place Order window to place a new order. This time we used ROLLBACK as the User Name (see Figure 12-22 on page 490). In the daisy chain scenario, this exception takes place after the insertions in the two DB2 tables and the update of the CICS VSAM file.

When the exception is thrown, the transaction is rolled back and the updates are backed out from:

- The ITSO.ORDER table
- The CICS VSAM file
- The ITSO.DISPATCH table

We can see in the WebSphere logs that the AtomicClient application successfully inserted the order record in the database prior to rolling back the transaction (Example 12-26).

Example 12-26 Catalog Application log

```
Catalogcontroller:doPost() - Action = Place Order
Catalogcontroller.handlePlaceOrder() - creating the order
Catalogcontroller.handlePlaceOrder()  beginning the transaction
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000524

```
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - simulating the RemoteException
CatalogController.handlePlaceOrder() - rollingback the transaction
```

We also can see that the DispatchOrderAtomic application successfully inserted the order record (Example 12-27).

Example 12-27 DispatchOrder Application log

```
DispatchOrderSoapBindingImpl.dispatchOrder(): ItemRef=10 Quantity=1
CustomerName=ROLLBACK Dept=Itso
DispatchOrderSoapBindingImpl.dispatchOrder() - creating the order
DispatchOrderSoapBindingImpl.dispatchOrder() - inserting the order in the
database
DispatchDB.insertDispatch() - inserted the order in the database!!!!
DispatchOrderSoapBindingImpl.dispatchOrder() - after the insert!!!
```

Following the RemoteException, the atomic transaction is rolled back. When we inquired on the stock level, we noted that there were still 30 items in stock.

12.4 Transaction scenario summary

These test scenarios demonstrate how you can synchronize WebSphere and CICS updates using WS-AT. They show how the classical 2PC distributed transaction can be implemented using Web services, and that the distributed global transaction can be committed or rolled back based on a set of Web service flows that are managed entirely by the WebSphere and CICS middleware.

Important: Before implementing a solution based on WS-AT, you should be aware of the general issues that can arise from any implementation of a distributed transaction (for example, locked records preventing access to important data) and you should also compare the solution with alternatives such as the J2EE Connector Architecture.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000525

Part 5

Appendixes

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000526

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000527

A

Sample handler programs

In this appendix we show the sample message handler and header processing programs used in our test scenarios:

- Sample message handler program - CIWSMSGH
- Sample header processing program - CIWSSECH
- Sample handler program - SNIFFER
- Sample XML parser program - MYPARSER
- Sample header processing program - CIWSSECR
- Sample header processing program - CIWSSECS
- Sample header processing program - WSATHND

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000528

A.1 Sample message handler program - CIWSMSGH

The CIWSMSGH program in Example A-1 was used to change the transaction ID for the Web service requests received in the pipeline.

The program obtains the value in the DFHFUNCTION container and if the program was invoked for a RECEIVE-REQUEST function, then it continues processing. It then obtains the data in the DFHWS-WEBSERVICE container and makes a decision based on the contents of the requested Web service name as to which transaction ID should be used. This new value for the transaction ID is put into the DFHWS-TRANID container. The program puts the contents of the DFHWS-TRANID and the DFHWS-WEBSERVICE containers to the CESE CICS transient data queue.

Example A-1 Sample message handler program - CIWSMSGH

```
CC01CC IDENTIFICATION DIVISION.
CC02CC PROGRAM-ID.  CIWSMSGH
CC03CC*****************************************************************
CC04CC*                                                               *
CC05CC* This program:                                                 *
CC06CC* 1. CHANGES THE DEFAULT TRANID : CPIH/CPIL                     *
CC07CC*    BASED ON THE INCOMING SOAP REQUESTS                        *
CC08CC*****************************************************************
CC09CC
CC10CC AUTHOR.      CHEREGATTI.
CC11CC DATE-COMPILED.
CC12CC ENVIRONMENT DIVISION.
CC13CC CONFIGURATION SECTION.
CC14CC SPECIAL-NAMES.
CC15CC DATA DIVISION.
CC16CC WORKING STORAGE SECTION.
CC17CC 01 WS-START.
CC18CC* Nesting offset for DISPLAYS
CC19CC   03 NN                    PIC X(11) VALUE 'CIWSMSGH: '.
CC20CC
CC21CC   03 RESP                     PIC S9(8) COMP-5 SYNC.
CC22CC   03 RESP2                    PIC S9(8) COMP-5 SYNC.
CC23CC   03 HOH-PTR                  USAGE IS POINTER.
CC24CC   03 HOH-LEN                  PIC S9(8) COMP-4.
CC25CC   03 CONTAINER-LEN            PIC S9(8) BINARY.
CC26CC   03 GETMAIN-PTR              USAGE IS POINTER.
CC27CC   03 GETMAIN-LEN              PIC S9(8) COMP-4.
CC28CC   03 WS-FAULT-STRING          PIC X(21) value spaces.
CC29CC   03 WS-DFHFUNCTION           PIC X(16) value spaces.
CC30CC   03 WS-WEBSERVICES           PIC X(32) value spaces.
CC31CC   03 WS-WEBSERVICES-LEN       PIC S9(8) BINARY
CC32CC*
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000529

```
CC33OC* Not found SOAP Fault Detail section
CC34OC*
CC35OC 01  WS-Fault-NotFnd.
CC36OC   03 WS-Fault-Namespace   pic x(53).
CC37OC   03 WS-Fault-OC-Lit      pic X(7).
CC38OC   03 WS-Fault-OC          pic X(2).
CC39OC   03 WS-Fault-RC-Lit      pic X(10).
CC40OC   03 WS-Fault-Item-Lit    pic X(11).
CC41OC   03 WS-Fault-Item        pic X(4).
CC42OC   03 WS-Fault-Item-Lit    pic X(12).
CC43OC   03 WS-Fault-CNamespace  pic x(19).
CC44OC
CC45OC 01 EXP4RSER-COM_LEN            PIC S9(4) COMP-4
CC46OC 01 C4-PARSER-MSG.
CC47OC    03 C4-PARSER-REQUEST.
CC48OC      05 FILLER                PIC X(2).
CC49OC      05 C4-TRANID             PIC X(4).
CC50OC    03 C4-PARSER-RESP-REQ      pic x(4).
CC51OC    03 C4 PARSER RET CODE      pic x(2).
CC52OC    03 FILLER                  PIC X(200).
CC53OC
CC54OC*****************************************************************
CC55OC* Externally referenced data areas
CC56OC*****************************************************************
CC57OC LINKAGE SECTION.
CC58OC 01 BIG-AREA.
CC59OC    02 FILLER    PIC X OCCURS 64000 DEPENDING ON BIG-LEN.
CC60OC
CC61OC 01 GETMAIN AREA.
CC62OC    02 FILLER    PIC X OCCURS 64000 DEPENDING ON GETMAIN LEN.
CC63OC 01 CONTAINER-DATA.
CC64OC    05 FILLER            PIC X OCCURS 32768
CC65OC                           DEPENDING ON CONTAINER-LEN.
CC66OC*----------------------------------------------------------------*
CC67OC    EJECT
CC68OC*----------------------------------------------------------------*
CC69OC PROCEDURE DIVISION.
CC70OC*                                                                *
CC71OC     EXEC CICS GET CONTAINER('DFHFUNCTION')
CC72OC                   INTO(WS-DFHFUNCTION)
CC73OC                   FLENGTH(length of WS-DFHFUNCTION)
CC74OC                   NOHANDLE
CC75OC     END-EXEC.
CC76OC* If not RECEIVE-REQUEST then exit
CC77OC     IF WS-DFHFUNCTION not = 'RECEIVE-REQUEST
CC78OC         PERFORM VALIDATE-REQUEST THRU END-VALIDATE-REQUEST
CC79OC         PERFORM CHANGE-TRANID    THRU END-CHANGE-TRANID
CC80OC         EXEC CICS
CC81OC            DELETE CONTAINER('DFHRESPONSE')
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000530

```
CC82CC         END-EXEC
CC83CC     END-IF
CC84CC     EXEC CICS RETURN END-EXEC.
CC85CC     GOBACK.
CC86CC*---------------------------------------------------------------
CC87CC     EJECT
CC88CC*---------------------------------------------------------------
CC89CC*              REQUEST VALIDATION
CC90CC*
CC91CC VALIDATE-REQUEST.
CC92CC     PERFORM GET-SOAP-WEBSERVICE THRU END-GET-SOAP-WEBSERVICE.
CC93CC     IF WS-WEBSERVICES = 'ERROR'
CC94CC         PERFORM FAULT-INVREQ
CC95CC     END-IF.
CC96CC END-VALIDATE-REQUEST. EXIT.
CC97CC*---------------------------------------------------------------
CC98CC     EJECT
CC99CC*---------------------------------------------------------------
C10CCC* Retrieve the C-HWS WEBSERVICE that contains type of requests
C101CC*---------------------------------------------------------------
C102CC GET-SOAP-WEBSERVICE.
C103CC     EXEC CICS
C104CC     GET CONTAINER('DFHWS-WEBSERVICE')
C105CC     SET(ADDRESS OF CONTAINER-DATA)
C106CC     FLENGTH(CONTAINER-LEN)
C107CC     END-EXEC.
C108CC* Copy the input container to our storage
C109CC     MOVE CONTAINER-DATA(1:50) TO WS-WEBSERVICES.
C110CC     MOVE CONTAINER-LEN TO WS-WEBSERVICES-LEN.
C111CC END-GET-SOAP-WEBSERVICE. EXIT.
C112CC*---------------------------------------------------------------
C113CC     EJECT
C114CC*---------------- CHANGE DEFAULT TRANID OPEN/CICS ---------------
C115CC CHANGE-TRANID.
C116CC       EXEC CICS GET CONTAINER('DFHWS-TRANID')
C117CC            SET(ADDRESS OF CONTAINER-DATA)
C118CC            FLENGTH(CONTAINER-LEN)
C119CC       END EXEC.
C120CC     IF WS-WEBSERVICES = 'inquireSingle'
C121CC        MOVE 'INQS' TO CA-TRANID
C122CC        PERFORM CHANGE-CONTAINER THRU END-CHANGE-CONTAINER
C123CC     END-IF
C124CC     IF WS-WEBSERVICES = 'inquireCatalog'
C125CC        MOVE 'INQC' TO CA-TRANID
C126CC        PERFORM CHANGE-CONTAINER THRU END-CHANGE-CONTAINER
C127CC     END-IF
C128CC     IF WS-WEBSERVICES   'placeOrder'
C129CC        MOVE 'ORDR' TO CA-TRANID
C130CC     END IF.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000531

```
C1291C     IF WS-WEBSERVICES = 'dispatch0-devEndpoint'
C1292C        MOVE 'DISP' TO CA-TRANID
C1293C        PERFORM CHANGE-CONTAINER THRU END-CHANGE-CONTAINER
C1310C     END-IF.
C1320C END-CHANGE-TRANID. EXIT.
C1330C*---------------------------------------------------------------
C1340C     EJECT
C1350C*
C1360C CHANGE CONTAINER.
C1370C     MOVE CA-TRANID TO CONTAINER-DATA(1:4)
C1380C     EXEC CICS PUT CONTAINER('DFHWS-TRANID')
C1390C          FROM(CONTAINER-DATA)
C1400C          FLENGTH(CONTAINER-LEN)
C1410C     END-EXEC.
C1420C     DISPLAY MM '+--------------------------------+'
C1430C     DISPLAY MM 'Container Name: : DFHWS-WEBSERVICE '.
C1440C*    DISPLAY MM 'Content length: '    WS-WEBSERVICES-LEN.
C1450C     DISPLAY MM 'Container contents: ' WS-WEBSERVICES.
C1460C     DISPLAY MM '                    '.
C1470C     DISPLAY MM 'Container Name: : DFHWS-TRANID  .
C1480C*    DISPLAY MM 'Content length: '    CONTAINER-LEN
C1490C     DISPLAY MM 'Container content: ' CONTAINER-DATA.
C1500C END-CHANGE-CONTAINER  EXIT.
C1510C     EJECT
C1520C*****************************************************************
C1530C* We detected that the ca request id field specifies an invalid
C1540C* request. This is a CLIENT error.
C1550C*****************************************************************
C1560C FAULT_INVREQ SECTION.
C1570C*                                                               *
C1580C* Generate a SOAP Fault
C1590C*---------------------------------------------------------------*
C1600C     MOVE 'Request code invalid' TO WS-FAULT-STRING
C1610C     EXEC CICS SOAPFAULT CREATE
C1620C               FAULTCODE(d'invalue|CLIENT|)
C1630C               FAULTSTRING(WS-FAULT-STRING)
C1640C               FAULTSTRLEN(length of WS-FAULT-STRING)
C1650C     END EXEC.
C1660C FAULT_INVREQ_END. EXIT.
C1670C*****************************************************************
C1680C* The supplied ca_item_req_ref reference is not in our database
C1690C* We decide to send a SOAP fault.
C1700C* This is a SERVER error
C1710C* We supply detailed information in the DETAIL fault element
C1720C*****************************************************************
C1730C FAULT-NOTFND SECTION.
C1740C*---------------------------------------------------------------*
C1750C* Build the Detail section.
C1760C*  we do it this way for pedagogical reasons
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000532

```
C1770C*-------------------------------------------------------------*
C1780C     MOVE '<o(   NOT FOUND   )o>' to WS-FAULT-STRING
C1790C     MOVE
C1800C     '<hdlh:FaultData ' xmlns:hdlh="http://HDR40LR2.fault">'
C1810C                              to WS-Fault-Namespace.
C1820C     MOVE '<hdlh:ReturnCode>' to WS-Fault-RC-Lit.
C1830C     MOVE DFHPARSER-RETURNCODE to WS-Fault-RC.
C1840C     MOVE '</hdlh:ReturnCode>' to WS-Fault-RC-Lit.
C1850C     MOVE '<hdlh:Item>'        to WS-Fault-Item-Lit.
C1860C     MOVE DFHPARSER-REF-REQ    to WS-Fault-Item.
C1870C     MOVE '</hdlh:Item>'       to WS-Fault-Item-Lit.
C1880C     MOVE '</hdlh:FaultDetail>'
C1890C            to WS-Fault-ENamespace.
C1900C     EXEC CICS SOAPFAULT CREATE
C1910C               DETAIL(WS-Fault-NotFnd)
C1920C               DETAILLENGTH(length of WS-Fault-NotFnd)
C1930C               FAULTCODE(DFHVALUE(SERVER))
C1940C               FAULTSTRING(WS-FAULT-STRING)
C1950C               FAULTSTRLEN(length of WS-FAULT-STRING)
C1960C     END-EXEC.
C1970C FAULT-INVREQ-EXC. EXIT.
```

Sample output from CIWSMSGH

Example A-2 shows the output sent to the CESE transient data queue by the CIWSMSGH message handler program, whenever it changes the transaction ID in the DFHWS-TRANID container.

Example A-2 CIWSMSGH - sample output

```
CIWSMSGH:  ----------------------------------
CIWSMSGH:  Container Name: - DFHWS-WEBSERVICE
CIWSMSGH:  Container Content- InquireCatalog
CIWSMSGH:  ----------------------------------
CIWSMSGH:  Container Name: - DFHWS-TRANID
CIWSMSGH:  Container Content: CIQC
CIWSMSGH:  >                                <
CIWSMSGH:  Container Name: - DFHWS-WEBSERVICE
CIWSMSGH:  Container Content- InquireSingle
CIWSMSGH:
CIWSMSGH:  Container Name: - DFHWS-TRANID
CIWSMSGH:  Container Content: CIQS
CIWSMSGH:  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
CIWSMSGH:  Container Name: - DFHWS-WEBSERVICE
CIWSMSGH:  Container Content- placeOrder
CIWSMSGH:  ----------------------------------
CIWSMSGH:  Container Name: - DFHWS-TRANID
CIWSMSGH:  Container Content- ORDR
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000533

A.2 Sample header processing program - CIWSSECH

The CIWSSECH program (Example A-3) extracts security credentials from the WS-Security header, validates them and changes the user ID under which the business logic of the invoked Web service executes

- It first determines that it is invoked during the RECEIVE-REQUEST phase of processing.
- It then ensures that this is only executed for a "place an order" request by examining the data in the DFHWS-URI container. In the event that this is not the case, it returns control to the calling program.
- When it is processing an order request, the WS-Security header is obtained from the DFHHEADER container.
- The header is then passed in a COMMAREA to a COBOL XML parsing program that extracts the user ID and password from the XML.
- CIWSSECH then uses the EXEC CICS VERIFY PASSWORD command to determine if the user ID and password are correct. If this is true, then the user ID is written to the DFHWS-USERID container, which will cause a context switch to occur when invoking the business logic. If the verification returns an error, then a SOAP fault is created and returned to the requester.

Example. A-3 Sample header processing program: CIWSSECH

```
CC01CC  PROCESS CICS
CC02CC IDENTIFICATION DIVISION.
CC03CC PROGRAM-ID.  CIWSSECH
CC04CC*****************************************************************
CC05CC*                                                               *
CC06CC* This program:                                                 *
CC07CC* 1. Checks for RECEIVE REQUEST function, else returns          *
CC08CC* 2. Obtains data from DFHHEADER container                       *
CC09CC* 3. Uses the PYPARSER program to extract the CA-USER-ID        *
CC10CC*                                    and the CA-PASSWORD        *
CC11CC*    fields from the XML message contained within the Body      *
CC12CC* 4. Verifies the user ID/password combination                   *
CC13CC*    If it is successful, then it places the CA-USER-ID into    *
CC14CC*        the DFHWS-USERID container.                            *
CC15CC*    If it fails, it returns a SOAP fault message in DFHHESHDSER*
CC16CC*****************************************************************
CC17CC
CC18CC AUTHOR.      CICSML.
CC19CC DATE-COMPILED.
CC20CC ENVIRONMENT DIVISION.
CC21CC CONFIGURATION SECTION
CC22CC SPECIAL-NAMES.
CC23CC DATA DIVISION.
CC24CC WORKING-STORAGE SECTION.
CC25CC 01 WS-STUFF.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000534

```
002600   03 FILLER                  PIC X(44)
002700       VALUE '*** CLASSIC WORKING STORAGE STARTS HERE ***'.
002800   03 WS-RESP                  PIC S9(8) COMP-5 SYNC.
002900   03 WS-RESP2                 PIC S9(8) COMP-5 SYNC.
003000   03 WS-HEAD-PTR              USAGE IS POINTER.
003100   03 WS-HEAD-LEN              PIC S9(8) COMP-4.
003200   03 WS-URI-PTR               USAGE IS POINTER.
003300   03 WS URI LEN               PIC S9(8) COMP 4.
003400   03 WS FUNC LEN              PIC S9(8) COMP 4 VALUE 16.
003500   03 WS GETPATH PTR           USAGE IS POINTER.
003600   03 WS-GETPATH-LEN           PIC S9(8) COMP-4.
003700   03 WS-FAULT-STRING          PIC X(40) value spaces.
003800   03 WS-FAULT-CODE            PIC S9(8) COMP-4
003900   03 WS-SOAP-LEVEL            PIC S9(8) COMP-4.
004000     88 WS-SOAP-11  VALUE 1.
004100     88 WS-SOAP-12  VALUE 2.
004200     88 WS-NOT-SOAP VALUE 10.
004300   03 WS-FUNC-AREA             PIC X(16).
004400   03 WS NOT AUTH              PIC X(40)
004500               VALUE  Not authorized to place order.'.
004600   03 WS-AUTH-FAILED           PIC X(40)
004700               VALUE  Authorization failed for order request.'.
004800
004900 01 MYPARSER-COM_LEN            PIC S9(4) COMP-4
005000 01 DFHPARSER-OSF.
005100    03 DFH-OSFH-ID             PIC X(8).
005200    03 DFH-PASSWORD            PIC X(8).
005300
005400 LINKAGE SECTION.
005500 01 WS HEAD AREA.
005600    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-HEAD-LEN.
005700
005800 01 WS-URI-AREA.
005900    02 FILLER    PIC X OCCURS 256 DEPENDING ON WS-URI-LEN.
006000
006100 01 WS-GETPATH-AREA.
006200    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-GETPATH-LEN.
006300
006400*****************************************************************
006500* Main line code begins                                         *
006600*****************************************************************
006700 PROCEDURE DIVISION.
006800 MAIN-PROCESSING SECTION.
006900
007000*****************************************
007100*  Receive the SOAP Body namespace
007200*****************************************
007300     PERFORM GET-DFHFUNCTION.
007400
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000535

```
007500*********************************
007600*  Receive the SOAP Body namespace
007700*********************************
007800     PERFORM GET-DFHWS-URI.
007900     IF WS-URI-AREA(1:WS-URI-LEN)
008000              NOT =  '/example4pp/placeOrder'
008100        EXEC CICS RETURN END-EXEC
008200     END-IF.
008300
008400*********************************
008500*  Receive the SOAP Body namespace
008600*********************************
008700     PERFORM GET-SOAP-HEADER.
008800
008900*****************************************************************
009000*  The SOAP body XML data can now be parsed by MYPARSER
009100*    LINK on the XML parser, it will return the CA-USER-ID
009700*                                          and CA-PASSWORD
009200*                                          or   'EXCEPTION'
009400*                                          or   'NOT FOUND'
009500*                                          or   'BAD COMMA'
009600*****************************************************************
009700     EXEC CICS LINK PROGRAM('MYPARSER')
009800                    COMMAREA(WS-HEAD-AREA)
009900                    LENGTH(MYPARSER-COMPLEN)
010000     END-EXEC.
010100     IF WS-HEAD-AREA(1:9)   'EXCEPTION' or
010700                            'NOT FOUND' or
010200                            'BAD COMMA'
010400*****************************************************
010500*  Error found during the XML PARSE program execution
010600*****************************************************
010700         MOVE WS-ALTH-FAILED TO WS-FAULT-STRING
010800         PERFORM FAULT-MESSAGE
010900     ELSE
011000         MOVE WS-HEAD-AREA to CA-PARSER-RESP
011100*****************************************
011200*  OK, data found, so do the VERIFY
011300*****************************************
011400         EXEC CICS VERIFY
011500            PASSWORD(CA-PASSWORD)
011600            USERID(CA-USER-ID)
011700            RESP(WS-RESP)
011800         END-EXEC
011900*****************
012000*  Successful V
012100*****************
012700         IF WS-RESP   DFHRESP(NORMAL)
012200            MOVE CA-  USER-ID
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000536

```
C12400         ELSE
C12500            MOVE WS-NOT-AUTH TO WS-FAULT-STRING
C12600            PERFORM FAULT-MESSAGE
C12700         END-IF
C12800     END-IF.
C12900*************************
C13000* and is the end, bye bye
C13100*************************
C13200     EXEC CICS RETURN END EXEC.
C13300
C13400 MAIN-PROCESSING-END. EXIT.
C13500
C13600*****************************************************
C13700*     ***********************************     *
C13800*              SUBROUTINES FOLLOW                 *
C13900*     ***********************************     *
C14000*****************************************************
C14100
C14200*****************************************************************
C14300* Retrieve the function type from DFHFUNCTION container
C14400*****************************************************************
C14500 GET-DFHFUNCTION.
C14600     EXEC CICS
C14700        GET CONTAINER( DFHFUNCTION')
C14800        INTO(WS-FUNC-AREA)
C14900        FLENGTH(WS-FUNC-LEN)
C15000        RESP(WS-RESP)
C15100     END-EXEC.
C15200
C15300*************************************************
C15400* Check for correct length. Create SOAP fault if
C15500* not correct. If for incoming request then we
C15600* do the work, else we exit immediately.
C15700*************************************************
C15800     IF WS-FUNC- Eh NOT - 15
C15900        MOVE WS-ALTH-FAILED TO WS-FAULT-STRING
C16000        PERFORM FAULT-MESSAGE
C16100     ELSE
C16200        IF WS FUNC AREA NOT = 'RECEIVE REQUEST'
C16300           EXEC CICS RETURN END EXEC
C16400        END-IF
C16500     END-IF.
C16600
C16700*****************************************************************
C16800* Retrieve the URI from the DFHWS-URI container
C16900*****************************************************************
C17000 GET-DFHWS-URI.
C17100     EXEC CICS
C17200        GET CONTAINER( DFHWS URI')
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000537

```
C17300        SET(WS-URI-PTR)
C17400        FLENGTH(WS-URI-LEN)
C17500        RESP(WS-RESP)
C17600     END-EXEC.
C17700
C17800*****************************************
C17900* Copy the input container to our storage
C18000*****************************************
C18100     IF WS URI LEN > 0
C18200        SET ADDRESS OF WS-REQ-AREA TO WS-URI-PTR
C18300        MOVE WS-URI-LEN TO WS-GETMAIN-LEN
C18400
C18500        EXEC CICS GETMAIN
C18600            SET(WS-GETMAIN-PTR)
C18700            FLENGTH(WS-GETMAIN-LEN)
C18800        END-EXEC
C18900
C19000        SET ADDRESS OF WS-GETMAIN-AREA TO WS-GETMAIN-PTR
C19100        MOVE WS URI AREA TO WS GETMAIN AREA
C19200        SET WS-URI-PTR TO WS-GETMAIN-PTR
C19300        SET ADDRESS OF WS-REQ-AREA TO WS-URI-PTR
C19400     ELSE
C19500        MOVE WS-PATH-FAILED TO WS-FAULT-STRING
C19600        PERFORM FAULT-MESSAGE
C19700        EXEC CICS RETURN END-EXEC
C19800     END-IF.
C19900
C20000*****************************************
C20100* Retrieve the <wsse:Security> header
C20200*****************************************
C20300 GET-SOAP-HEADER.
C20400     EXEC CICS
C20500        GET CONTAINER( OFFHEADER')
C20600        SET(WS-HEAD-PTR)
C20700        FLENGTH(WS-HEAD-LEN)
C20800        RESP(WS-RESP)
C20900     END-EXEC.
C21000
C21100*****************************************
C21200* Copy the input container to our storage
C21300*****************************************
C21400     IF WS-HEAD-LEN > 0
C21500        SET ADDRESS OF WS-HEAD-AREA TO WS-HEAD-PTR
C21600        MOVE WS-HEAD-LEN TO WS-GETMAIN-LEN MYPARSER-CONLEN
C21700
C21800        EXEC CICS GETMAIN
C21900            SET(WS-GETMAIN-PTR)
C22000            FLENGTH(WS-GETMAIN-LEN)
C22100        END EXEC
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000538

```
C2220C
C2230C        SET ADDRESS OF WS-GETMAIN-AREA TO WS-GETMAIN-PTR
C2240C        MOVE WS-HEAD-AREA TO WS-GETMAIN-AREA
C2250C        SET WS-HEAD-PTR TO WS-GETMAIN-PTR
C2260C        SET ADDRESS OF WS-HEAD-AREA TO WS-HEAD-PTR
C2270C     ELSE
C2280C        MOVE WS-ALTE-FAILED TO WS-FAULT-STRING
C2290C        PERFORM FAULT-MESSAGE
C2300C        EXEC CICS RETURN END EXEC
C2310C     END-IF.
C2320C
C2330C*****************************************************************
C2340C* The user ID has been verified, we will now set the contents of
C2350C* DFHWS-USERID container with this value.This will cause the
C2360C* business logic (back-end program/s) to be executed with this
C2370C* user ID, using it's access rights.
C2380C*****************************************************************
C2390C SET-USER-ID.
C2400C     EXEC CICS
C2410C         PUT CONTAINER('DFHWS-USERID')
C2420C         FROM(CA-USER-ID)
C2430C         FLENGTH(length of CA-USER-ID)
C2440C         DATATYPE(DFHVALUE(CHAR))
C2450C         RESP(WS-RESP)
C2460C     END-EXEC.
C2470C
C2480C*****************************************************************
C2490C* We detected that the ca_request_id field specifies an invalid
C2500C* request. This is a CLIENT error.
C2510C*****************************************************************
C2520C FAULT-MESSAGE SECTION.
C2530C************************
C2540C* Generate a SOAP Fault
C2550C************************
C2560C     EXEC CICS
C2570C         GET CONTAINER('DFHWS-SOAPLEVEL')
C2580C         INTO(WS-SOAP-LEVEL)
C2590C         FLENGTH(WS-READ-LEN)
C2600C         RESP(WS-RESP)
C2610C     END-EXEC.
C2620C
C2630C*************************************
C2640C* MOVE CORRECT VERSION OF FAULTCODE
C2650C*************************************
C2660C     IF WS-SOAP-11 MOVE DFHVALUE(CLIENT) TO WS-FAULT-CODE
C2670C     ELSE
C2680C        MOVE DFHVALUE(SENDER) TO WS-FAULT-CODE
C2690C     END-IF
C2700C
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000539

```
027100     EXEC CICS SCAFFAULT CREATE
027200               FAULTSTRING(WS-FAULT-STRING)
027300               FAULTSTRLEN(LENGTH OF WS-FAULT-STRING)
027400               FAULTCODE(WS-FAULT-CODE)
027500     END-EXEC.
027600 FAULT-MESSAGE-END. EXIT.
```

A.3 Sample handler program - SNIFFER

The SNIFFER handler program can be used as a message handler program or a header processing program.

It is a very simple program that browses through the containers available in the pipeline. It does this by issuing a STARTBROWSE CONTAINER command followed by GETNEXT CONTAINER until all have been browsed, then it issues an ENDBROWSE CONTAINER command. For each container that it issues a GETNEXT CONTAINER for, it displays the name and contents to the CICS transient data queue CESE

Example A-4 Sample handler program - SNIFFER

```
000100CBL TRUNC(OPT)
000200 IDENTIFICATION DIVISION.
000300 PROGRAM-ID. SNIFFER.
000400*****************************************************************
000500* HEADER START                                                  *
000600*                                                               *
000700*                                                               *
000800* PROGRAM NAME: See above                                       *
000900*                                                               *
001000* TITLE: Pipeline Channel container sniffer                     *
001100*                                                               *
001200* DATE: 03/06/05                                                *
001300*                                                               *
001400* AUTHOR: Ian Noble                                             *
001500*                                                               *
001600* CHANGE HISTORY:                                               *
001700*                                                               *
001800*   DATE MODIFIED  CHANGED BY        REASON FOR CHANGE          *
001900*                                                               *
002000* PROGRAM DESCRIPTION.                                          *
002100*                                                               *
002200*   This program writes a report to the CICS transient data     *
002300*   queue which is usually an extra-partition queue having a    *
002400*   DDname of CEEMSG. The report contains the name, length, and *
002500*   contents of every container which is visible to user code   *
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000540

```
CC26CC*  within a pipeline as shown in the following example:          *
CC27CC*                                                               *
CC28CC*  SNIFFER :  *** Start ***                                     *
CC29CC*  SNIFFER :  ********************************                 *
CC30CC*  SNIFFER :  Container Name   : DFHFUNCTION                    *
CC31CC*  SNIFFER :  Content length   : 00000016                       *
CC32CC*  SNIFFER :  Container content:                                *
CC33CC*  RECEIVE HEQLEST                                             *
CC34CC*  SNIFFER :  Containers on channel: List starts.               *
CC35CC*  SNIFFER :  >-------------------------------<                 *
CC36CC*  SNIFFER :  Container Name   : DFH-HANDLEPLIST                *
CC37CC*  SNIFFER :  Content length   : 00000000                       *
CC38CC*  SNIFFER :  Container EMPTY                                   *
CC39CC*  SNIFFER :  >-------------------------------<                 *
CC40CC*  SNIFFER :  Container Name   : DFHNFSFCh$F                    *
CC41CC*  SNIFFER :  Content length   : 00000000                       *
CC42CC*  SNIFFER :  Container EMPTY                                   *
CC43CC*  SNIFFER :  >                               <                 *
CC44CC*  SNIFFER :  Container Name   : UFHFUNCTION                    *
CC45CC*  SNIFFER :  Content length   : 00000016                       *
CC46CC*  SNIFFER :  Container content:                                *
CC47CC*  RECEIVE-REQUEST                                             *
CC48CC*  SNIFFER :  ********************************                 *
CC49CC*  SNIFFER :  Container Name   : DFHWS-SOAPACTION               *
CC50CC*  SNIFFER :  Content length   : 00000007                       *
CC51CC*  SNIFFER :  Container content:                                *
CC52CC*  ""                                                          *
CC53CC*  SNIFFER :  >                                   <             *
CC54CC*  SNIFFER :  Container Name   : UFHWS URL                      *
CC55CC*  SNIFFER :  Content length   : 00000022                       *
CC56CC*  SNIFFER :  Container content:                                *
CC57CC*  /exampleApp/placeOrder                                      *
CC58CC*  SNIFFER :  ********************************                 *
CC59CC*  SNIFFER :  Container Name   : DFHREQLIST                     *
CC60CC*  SNIFFER :  Content length   : 00007C94                       *
CC61CC*  SNIFFER :  Container content:                                *
CC62CC*  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoa...    *
CC63CC*  " xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:x...    *
CC64CC*  lsoap.org/ws/2004/10/wsaddr" xmlns:wsa="http://schemas...    *
CC65CC*  etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc.      *
CC66CC*  SNIFFER :  ********************************                 *
CC67CC*  SNIFFER :  Container Name   : DFH-SERVICEPLIST               *
CC68CC*  SNIFFER :  Content length   : 00000134                       *
CC69CC*  SNIFFER :  Container content:                                *
CC70CC*  <registration service endpoint>       http://MWS33.top etc.  *
CC71CC*  SNIFFER :  >-------------------------------<                 *
CC72CC*  SNIFFER :  Container Name   : UFHWS-PIPELINE                 *
CC73CC*  SNIFFER :  Content length   : 00000008                       *
CC74CC*  SNIFFER :  Container content:                                *
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000541

```
CC75CC*   PIPEL                                                          *
CC76CC*   SNIFFER :  ********************************                  *
CC77CC*   SNIFFER :  Container Name   : DFHWS-USERID                    *
CC78CC*   SNIFFER :  Content length   : 00000008                        *
CC79CC*   SNIFFER :  Container content:                                 *
CC80CC*   CICS3D                                                       *
CC81CC*   SNIFFER :  >                                 <               *
CC82CC*   SNIFFER :  Container Name   : DFHWS-TRANID                   *
CC83CC*   SNIFFER :  Content length   : 00000004                        *
CC84CC*   SNIFFER :  Container content:                                 *
CC85CC*   IPIC                                                         *
CC86CC*   SNIFFER :  ********************************                  *
CC87CC*   SNIFFER :  Container Name   : DFHWS-ACCESSACCE               *
CC88CC*   SNIFFER :  Content length   : 00000C32                        *
CC89CC*   SNIFFER :  Container content:                                 *
CC90CC*   placeOrder                                                   *
CC91CC*   SNIFFER :  >                                 <               *
CC92CC*   SNIFFER :  Container Name   : DFHWS-APPHANDLER                *
CC93CC*   SNIFFER :  Content length   : 00000008                        *
CC94CC*   SNIFFER :  Container content:                                 *
CC95CC*   DFHPITP                                                       *
CC96CC*   SNIFFER :  Containers on channel: List ends                  *
CC97CC*   SNIFFER :  DFHRESPONSE      Container deleted                 *
CC98CC*   SNIFFER :  **** End ****                                     *
CC99CC*                                                                 *
CCCCCC*-------------------------------------------------------------------*
C10010C
C10020C ENVIRONMENT DIVISION.
C10030C DATA DIVISION.
C10040C WORKING STORAGE SECTION.
C10050C
C10060C 77  A5-EYE-CATCHER         PIC X(44)     VALUE
C10070C                  '*** SNIFFER WORKING STORAGE STARTS HERE ***'.
C10080C
C10090C* Working offset for DISPLAYS
C10100C 77  WW                    PIC X(11) VALUE 'SNIFFER : '.
C10110C
C10120C* Miscellaneous variables
C10130C
C10140C 77 CONTAINER-FOUND         PIC X(01).
C10150C
C10160C 77 CHANNEL-NAME               PIC X(16).
C10170C 77 CONTAINER-NAME             PIC X(16).
C10180C 77 CONTAINER-LEN              PIC S9(8) COMP-4 SYNCHRONIZED
C10190C                                          VALUE 0.
C10200C
C10210C 77 DFHEIBLK-ITEM-PTR          USAGE IS POINTER.
C10220C 77 DFHEIBLK-ITEM-LEN          PIC S9(8) COMP-4 SYNCHRONIZED
C10230C                                          VALUE 0.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000542

```
012400
012500 77 GET-MAIN-PTR                USAGE IS POINTER
012600 77 GET-MAIN-LEN                PIC S9(8) COMP-4 SYNCHRONIZED
012700                                          VALUE 0.
012800
012900* DFH STRUC declarations
013000 COPY DFHAPIDC.
013100
013200 77 FUNCTION-HOLDER          PIC X(16).
013300
013400 77 CALL-LABEL                 PIC X(08).
013500 77 CALL-DETAILS               PIC X(80).
013600
013700 77 RESP                       PIC S9(8)  COMP-5 SYNC.
013800 77 RESP2                      PIC S9(8)  COMP-5 SYNC.
013900
014000* Channel browse stuff
014100 01 CHANNEL-BROWSE.
014200    03 BROWSE-TOKEN            PIC S9(8) COMP-4 SYNC.
014300    03 BROWSED-NAME            PIC X(16).
014400
014500*---------------------------------------------------------------*
014600* Constants
014700*---------------------------------------------------------------*
014800
014900 77 UNEXPECTED-RESP-ASCODE     PIC X(04) VALUE  SPACE.
015000
015100*---------------------------------------------------------------*
015200 LINKAGE SECTION.
015300
015400 01 DFHFUNCTION-IN.
015500    03 FILLER                   PIC X OCCURS 1 TO 54
015600                                DEPENDING ON DFHFUNCTION-IN-LEN.
015700
015800 01 GET-MAIN-AREA.
015900    03 FILLER                   PIC X OCCURS 1 TO 65536
016000                                      DEPENDING ON GET-MAIN-LEN.
016100
016200
016300*
016400 PROCEDURE DIVISION.
016500 MAIN-PROG SECTION.
016600
016700     Display MM 'MM Start ***'.
016800
016900     PERFORM GET-DFHFUNCTION-CONTAINER.
017000
017100     PERFORM BROWSE-ALL-CONTAINERS.
017200
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000543

```
C17300     IF FUNCTION-DRIVEN = PI-SEND-REQUEST
C17400     OR FUNCTION-DRIVEN = PI-RECEIVE-REQUEST
C17500     OR FUNCTION-DRIVEN = PI-PROCESS-REQUEST
C17600     THEN
C17700        PERFORM DEFINE-DEFRESPONSE
C17800     END-IF.
C17900
C18000     Display MM '**** End ****'.
C18100
C18200     EXEC CICS RETURN
C18300     END-EXEC.
C18400
C18500
C18600*****************************************************************
C18700*
C18800* Retrieve the contents of the DFHFUNCTION container
C18900*
C19000*****************************************************************
C19100 GET DFHFUNCTION CONTAINER.
C19200
C19300     MOVE PI-DFHFUNCTION TO CONTAINER-NAME.
C19400     PERFORM GET-NAMED-CONTAINER.
C19500
C19600     IF CONTAINER-FOUND = 'Y'
C19700     THEN
C19800        MOVE CONTAINER-LEN TO DFHFUNCTION-IN-LEN
C19900        SET DFHFUNCTION-IN-PTR TO GETDATA-PTR
C20000        SET ADDRESS OF DFHFUNCTION-IN TO DFHFUNCTION-IN-PTR
C20100        MOVE DFH-FUNCTION-IN TO FUNCTION-DRIVEN
C20200     END-IF.
C20300
C20400
C20500*****************************************************************
C20600*
C20700* Retrieve a named container
C20800*
C20900*****************************************************************
C21000 GET NAMED CONTAINER.
C21100
C21200     MOVE 'N' TO CONTAINER-FOUND
C21300* Get the length of the name container from the input pipe.
C21400     MOVE 'SNIFFCCL' TO CALL-LABEL.
C21500     EXEC CICS GET
C21600               CONTAINER(CONTAINER-NAME)
C21700               NODATA
C21800               FLENGTH(CONTAINER-LEN)
C21900               NOHANDLE
C22000               RESP(RESP)
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000544

```
C2210C              RESP2(RESP2)
C2220C     END-EXEC.
C2230C
C2240C* If the container was't found, do nothing   Else GETMAIN
C2250C* suitable storage and retrieve it for real.
C2260C     EVALUATE RESP
C2270C        WHEN DFHRESP(CONTAINERERR)
C2280C           DISPLAY NN  Container NOT FOUND'
C2290C        WHEN DFHRESP(NORMAL)
C2300C           MOVE 'Y' TO CONTAINER FOUND
C2310C
C2320C           DISPLAY NN  ********************************'
C2330C           DISPLAY NN  Container Name   : 'CONTAINER-NAME
C2340C           DISPLAY NN  Content length   : ' CONTAINER-LEN
C2350C
C2360C           IF CONTAINER-LEN > 0
C2370C           THEN
C2380C              MOVE 'SNDFROUT' TO CALL-LABEL
C2390C              MOVE CONTAINER-LEN TO GETMAIN-LEN
C2400C              EXEC CICS GETMAIN SET(GETMAIN-PTR)
C2410C                                FLENGTH(GETMAIN-LEN)
C2420C              END-EXEC
C2430C              SET ADDRESS OF GETMAIN-AREA TO GETMAIN-PTR
C2440C              EXEC CICS GET CONTAINER(CONTAINER-NAME)
C2450C                           INTO(GETMAIN-AREA)
C2460C                           FLENGTH(CONTAINER-LEN)
C2470C              END-EXEC
C2480C              DISPLAY NN 'Container content.
C2490C                     GETMAIN AREA
C2500C
C2510C           ELSE
C2520C              DISPLAY NN 'Container EMPTY'
C2530C           END-IF
C2540C        WHEN OTHER
C2550C           DISPLAY NN  Error on GET CONTAINER   CONTAINER-NAME
C2560C           PERFORM GENERIC-ABEND
C2570C     END-EVALUATE.
C2580C
C2590C
C2600C*****************************************************************
C2610C*
C2620C* BROWSE all containers on the current Channel
C2630C*
C2640C*****************************************************************
C2650C BROWSE-ALL-CONTAINERS.
C2660C
C2670C     DISPLAY NN 'Containers on channel: .ist starts.'
C2680C
C2690C* Start the Channel Browse
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000545

```
C27COC     MOVE 'SNIFFCC1' TO CALL-LABEL.
C27DOC     EXEC CICS STARTBROWSE
C27EOC               CONTAINER
C27FOC               BROWSETOKEN(BROWSE-TOKEN)
C27GOC               RESP(RESP)
C27HOC               RESP2(RESP2)
C27IOC     END-EXEC.
C27JOC
C27KOC* Browse the next container name
C27LOC     PERFORM UNTIL RESP NOT = DFHRESP(NORMAL)
C27MOC
C27NOC        MOVE 'SNIFFCC4' TO CALL-LABEL
C27OOC        EXEC CICS GETNEXT
C27POC                  CONTAINER(BROWSED-NAME)
C27QOC                  BROWSETOKEN(BROWSE-TOKEN)
C27ROC                  RESP(RESP)
C27SOC                  RESP2(RESP2)
C27TOC        END-EXEC
C27UOC
C27VOC        IF RESP = DFHRESP(NORMAL)
C27WOC        THEN
C27XOC           MOVE BROWSED-NAME TO CONTAINER-NAME
C27YOC           PERFORM GET-NAMED-CONTAINER
C27ZOC        END-IF
C280OC
C281OC     END-PERFORM.
C282OC
C283OC     IF RESP NOT = DFHRESP(END)
C284OC     THEN
C285OC        PERFORM GENERIC-ABEND
C286OC     END-IF.
C287OC
C288OC     DISPLAY MM 'Containers on channel: list ends'
C289OC
C290OC* End the Channel Browse
C291OC     MOVE 'SNIFFCC5' TO CALL-LABEL.
C292OC     EXEC CICS ENDBROWSE
C293OC               CONTAINER
C294OC               BROWSE-TOKEN(BROWSE-TOKEN)
C295OC               RESP(RESP)
C296OC               RESP2(RESP2)
C297OC     END-EXEC.
C298OC     IF RESP NOT = DFHRESP(NORMAL)
C299OC     THEN
C300OC        PERFORM GENERIC-ABEND
C301OC     END-IF.
C302OC
C303OC
C304OC*****************************************************************
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000546

```
C31900*                                                                *
C32000* Delete the DFHRESPONSE container                               *
C32100*                                                                *
C32200******************************************************************
C32300 DELETE-DFHRESPONSE.
C32400
C32500     MOVE 'SNIFFCOB' TO CALL-LABEL.
C32600     EXEC CICS ASSIGN
C32700               CHANNEL(CHANNEL-NAME)
C32800               RES-(RES-)
C32900               RESP2(RESP2)
C33000     END-EXEC.
C33100
C33200     IF CHANNEL-NAME =  'DFHWEC-V1' THEN
C33300        DISPLAY NN ' >> a SOAP header processing program.....'
C33400     ELSE
C33500        EXEC CICS DELETE
C33600                  CONTAINER(PI-DFHRESPONSE)
C33700                  RESP(RESP)
C33800                  RESP2(RESP2)
C33900        END-EXEC
C34000        EVALUATE RESP
C34100           WHEN DFHRESP(NORMAL)
C34200              DISPLAY NN PI-DFHRESPONSE ' container deleted'
C34300           WHEN OTHER
C34400              PERFORM GENERIC-ABEND
C34500        END-EVALUATE
C34600     END-IF.
C34700
C34800
C34900******************************************************************
C35000*
C35100* Generic abend
C35200*
C35300******************************************************************
C35400 GENERIC-ABEND.
C35500
C35600
C35700      DISPLAY NN '***** Unable to continue ***** '.
C35800      DISPLAY NN 'Unexpected RESP code from EXEC CICS call'.
C35900      DISPLAY NN 'Call label : ' CALL-LABEL.
C36000      DISPLAY NN 'RESP  : ' RESP.
C36100      DISPLAY NN 'RESP2 : ' RESP2.
C36200      IF CALL-DETAILS NOT = SPACES
C36300      THEN
C36400         DISPLAY NN  'Further details:'
C36500                   CALL-DETAILS
C36600      END-IF.
C36700
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000547

```
C3680C      EXEC CICS ABEND
C3690C         ABCODE(UNEXPECTED-RESP-ABCODE)
C3700C      END-EXEC.
```

Sample output from the SNIFFER program

Example A-5 shows sample output from the SNIFFER program. The output is sent to CICS transient data queue CESE

Example A-5 SNIFFER program - sample output

```
SNIFFER :  *** Start ***
SNIFFER :  ********************************
SNIFFER :  Container Name   - DFHEJNCTION
SNIFFER :  Content length   - 00000016
SNIFFER :  Container content: RECEIVE-REQUEST
SNIFFER :  Containers on channel: list starts.
SNIFFER :  >                              <
SNIFFER :  Container Name   . DFH WRAPPLEHFLIST
SNIFFER :  Content length   . 00000000
SNIFFER :  Container EMPTY
SNIFFER :  ********************************
SNIFFER :  Container Name   : DFHRESPONSE
SNIFFER :  Content length   - 00000000
SNIFFER :  Container EMPTY
SNIFFER :  >------------------------------<
SNIFFER :  Container Name   : DFHEJNCTION
SNIFFER :  Content length   . 00000010
SNIFFER :  Container content: RECEIVE-REQUEST
SNIFFER :  >------------------------------<
SNIFFER :  Container Name   : DFHWS-SOAPACTION
SNIFFER :  Content length   : 00000002
SNIFFER :  Container content: ""
SNIFFER :  ********************************
SNIFFER :  Container Name   - DFHWS-URI
SNIFFER :  Content length   : 00000075
SNIFFER :  Container content: /exampleApp/inquireString +
SNIFFER :  >                              <
SNIFFER :  Container Name   . DFHREQUEST
SNIFFER :  Content length   . 00000307
SNIFFER :  Container content: <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http://
schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Header/><soapenv
:Body><m528:DFHCXCMH xmlns:m528="http://www.DFHCXCMH_DFHWS2LS_Request.com"><m528
:ca_req_str id>QIINQS</m528:ca_request_id><m528:ca_return_code>0</m528:ca_retur
n_code><m529:ca_response_message>
</m528:ca_response_message><m528:ca_inquire_single><m528:ca_user_ref_req>10</m5
28:ca_item_ref_req><m528:file></m528:filler> </m528:filler>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000548

```
<p628:ca_single_item><p628:ca_sngl_item_ref>0</p628:ca_sngl_item_ref><p628:ca_s
ngl_description>
</p628:ca_sngl_description><p628:ca_sngl_department>0</p628:ca_sngl_department>
<p628:ca_sngl_cost>0.0</p628:ca_sngl_cost><p628:in_sng_stock>0</p628:in_sng_s
tock><p628:on_sngl_order>0</p628:on_sngl_order></p628:ca_single_item></p628:ca
_inquire_single></p628:DFHCXCMD></soapenv:Body></soapenv:Envelope>
SNIFFER :  >                                 <
SNIFFER :  Container name   : DFH_SERVICEPLIST
SNIFFER :  Content length   : 00000000
SNIFFER :  Container EMPTY
SNIFFER :  <---------------------------------
SNIFFER :  Container name   : DFHWS-PIPELINE
SNIFFER :  Content length   : 00000008
SNIFFER :  Container content: EXPIPE01
SNIFFER :  >---------------------------------<
SNIFFER :  Container name   : DFHWS-JSFRIC
SNIFFER :  Content length   : 00000009
SNIFFER :  Container content: CICSSJ
SNIFFER :  <---------------------------------<
SNIFFER :  Container name   : DFHWS-TRANID
SNIFFER :  Content length   : 00000004
SNIFFER :  Container content: CPIH
SNIFFER :  >---------------------------------<
SNIFFER :  Container name   : DFHWS-WEBSERVICE
SNIFFER :  Content length   : 00000007
SNIFFER :  Container content: inquireSingle
SNIFFER :  >                                 <
SNIFFER :  Container name   : DFHWS-APPHANDLER
SNIFFER :  Content length   : 00000000
SNIFFER :  Container content: DFHPITP
SNIFFER :  Containers on channel: List ends
SNIFFER :  DFHRESPONSE      container deleted
SNIFFER :  **** End ****
CICSMSG1:  >---------------------------------<
CICSMSG1:  Container name: : DFHWS-WEBSERVICE
CICSMSG1:  Container content: inquireSingle
CICSMSG1:  ---------------------------------
CICSMSG1:  Container name: . DFHWS-TRANID
CICSMSG1:  Container content. CWQS
SNIFFER :  *** Start ***
SNIFFER :  <---------------------------------
SNIFFER :  Container name   : DFHFUNCTION
SNIFFER :  Content length   : 00000016
SNIFFER :  Container content: SEND-RESPONSE
SNIFFER :  Containers on channel: list starts.
SNIFFER :  >---------------------------------<
SNIFFER :  Container name   . DFHRESPONSE
SNIFFER :  Content length   . 00001047
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000549

```
SNIFFER :  Container content:
<SOAP-ENV:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmln
s:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.o
rg/2001/XMLSchema" xmlns:xsi ="http://www.w3.org/2001/XMLSchema-instance" xmlns:SO
AP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Body><DFHCICMN xmlns
:ns1="http://www.DFHCICMN.DFHJAVDP.Response.com"><ca_request_id>CITMQ</ca
_request_id><ca_return_code>0</ca_return_code><ca_response_message>RETURNED
ITEM. REF =0010
</ca_response_message><ca_inquire_single><ca_item_ref_req>10</ca_item_ref_req>
<filler1>    </filler1><filler2>
</filler2><ca_single_item><ca_sngl_item_ref>10</ca_sngl_item_ref><ca_sngl_descr
iption>Ball Pens Black2pc
</ca_sngl_description><ca_sngl_department>10</ca_sngl_department><ca_sngl_cost>
002.90</ca_sngl_cost><ca_sngl_qty_stock>0</in_sngl_stock><on_sngl_order>0</on_sngl
_order></ca_single_item></ca_inquire_single></DFHCICMN></SOAP-ENV:Body></
/SOAP-ENV:Envelope>
SNIFFER :  >                                 <
SNIFFER :  Container Name   : DFH-HANDLERLIST
SNIFFER :  Content length   : 00000000
SNIFFER :  Container EMPTY
SNIFFER :  <---------------------------------
SNIFFER :  Container Name   : DFHFUNCTION
SNIFFER :  Content length   : 00000016
SNIFFER :  Container content: SEND-RESPONSE
SNIFFER :  >---------------------------------<
SNIFFER :  Container Name   : DFHWS-SOAPACTION
SNIFFER :  Content length   : 00000002
SNIFFER :  Container content: ""
SNIFFER :  >---------------------------------<
SNIFFER :  Container Name   : DFHWS-URI
SNIFFER :  Content length   : 00000025
SNIFFER :  Container content: /exampleApp/inquireSingle
SNIFFER :  <---------------------------------
SNIFFER :  Container Name   : DFH-SERVICELIST
SNIFFER :  Content length   : 00000000
SNIFFER :  Container EMPTY
SNIFFER :  >                                 <
SNIFFER :  Container Name   : DFHWS-PIPELINE
SNIFFER :  Content length   : 00000009
SNIFFER :  Container content: SOAPIPEDI
SNIFFER :  <---------------------------------
SNIFFER :  Container Name   : DFHWS-USERID
SNIFFER :  Content length   : 00000008
SNIFFER :  Container content: CICSUSER
SNIFFER :  >---------------------------------<
SNIFFER :  Container Name   : DFHWS-TRANID
SNIFFER :  Content length   : 00000004
SNIFFER :  Container content: CWXN
SNIFFER :  >---------------------------------<
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000550

```
SNIFFER :  Container name   : DFHWS-WEBSERVICE
SNIFFER :  Content length   : 00000032
SNIFFER :  Container content: InquireSingle
SNIFFER :  ********************************
SNIFFER :  Container name   : DFHWS-APPHANDLER
SNIFFER :  Content length   : 00000009
SNIFFER :  Container content: DFHHTTP
SNIFFER :  ********************************
SNIFFER :  Container name   : DFHWS-OPERATION
SNIFFER :  Content length   : 00000009
SNIFFER :  Container content: DFHDORMH
SNIFFER :  Containers on channel: List ends
SNIFFER :  **** End ****
```

A.4 Sample XML parser program - MYPARSER

The MYPARSER program shown in Example A-6 is used by the CIWSSECH header processing program to parse the SOAP header in the DFHWS-DFHHEADER container. It uses the COBOL XML PARSE statement to parse through the SOAP WS-Security header that was received in the COMMAREA, extracting the user ID and password. This extracted data is then returned in the COMMAREA for the calling program to use.

Refer to the following publications for more information on XML parsing within COBOL programs:

- *Enterprise COBOL for z/OS V3R3 Programming Guide, SC27-1412*
- *Enterprise COBOL for z/OS V3R3 Language Reference, SC27-1408*

Example A-6 Sample XML parser program - MYPARSER

```
000100 PROCESS CICS
000200 IDENTIFICATION DIVISION.
000300 PROGRAM-ID.  MYPARSER
000400
000500 DATE COMPILED.
000600 ENVIRONMENT DIVISION.
000700 CONFIGURATION SECTION.
000800 SPECIAL-NAMES.
000900 DATA DIVISION.
001000 WORKING-STORAGE SECTION.
001100 01 WS-START.
001200    03 FILLER                    PIC X(44)
001300       VALUE '**** MYPARSER WORKING STORAGE STARTS HERE ****.
001400    03 RESP                      PIC S9(8) COMP-5 SYNC.
001500    03 RESP2                     PIC S9(8) COMP-5 SYNC.
001600    03 I                         PIC S9(4) COMP-5 SYNC.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000551

```
CC170C   03 J                          PIC S9(4) COMP-5 SYNC.
CC180C   03 K                          PIC S9(4) COMP-5 SYNC.
CC190C   03 L                          PIC S9(4) COMP-5 SYNC.
CC200C   03 WORKAREA                   PIC X(24).
CC210C*****************************************************
CC220C* Data items for use by parser
CC230C*****************************************************
CC240C   03 IN-ELEM                    PIC X(1) VALUE 'N'.
CC250C   03 IN-REF-REC                 PIC X(1) VALUE 'N'.
CC260C   03 PARSE-ERROR                PIC X(1) VALUE 'N'.
CC270C
CC280C* The XML document to parse
CC290C   01 XML-DOCUMENT PIC X(1024).
CC300C
CC310C   01 USERNAME-XML TAG-FOUND  PIC X VALUE 'N'.
CC320C
CC330C*****************************************************
CC340C* Externally referenced data areas
CC350C*****************************************************
CC360C LINKAGE SECTION.
CC370C 01  DFHCOMMAREA.
CC380C   02  DFHCOMMAREA-LN          PIC X(LC24).
CC390C   02  CA-RSP REDEFINES DFHCOMMAREA-LN.
CC400C     05   CA-USER-ID           PIC X(8).
CC410C     05   CA-PASSWORD          PIC X(8).
CC420C     05   FILLER               PIC X(1008).
CC430C
CC440C*****************************************************
CC450C* Main line code begins                             *
CC460C*****************************************************
CC470C PROCEDURE DIVISION.
CC480C MAIN-PROCESSING SECTION.
CC490C**************************
CC500C*    Validate the commarea
CC510C**************************
CC520C     PERFORM INIT-AND-VALIDATE.
CC530C*****************************************************
CC540C* Received a valid COMMAREA so invoke the XML parser
CC550C* Invoke the XML parser passing it the XML message
CC560C* What is the CICS command used to invoke the XML parser?
CC570C*****************************************************
CC580C     XML PARSE XML-DOCUMENT
CC590C        PROCESSING PROCEDURE XML-HANDLER
CC600C     END-XML.
CC610C
CC620C     PERFORM PERFORM-RETURN-RESPONSE.
CC630C
CC640C     EXEC CICS RETURN END-EXEC.
CC650C MAIN-PROCESSING-END. EXIT.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000552

```
CC66OC
CC67OC*****************************************************************
CC68OC* The following section is executed as a callback routine
CC69OC* out of the XML PARSE statement above.
CC70OC* What is the name of the register we need to check on the
CC71OC* EVALUATE statement?
CC72OC*****************************************************************
CC73OC XML-HANDLER SECTION.
CC74OC     EVALUATE XML-EVENT
CC75OC        WHEN 'START-OF-ELEMENT'
CC76OC*---------------------------------------
CC77OC* Check if we have an element of interest
CC78OC* i.e. Username and Password
CC79OC*---------------------------------------
CC80OC           IF XML-TEXT = 'wssp:Username'
CC81OC              MOVE 'Y' TO IN-ELEM USERNAME-XML-TAG-FOUND
CC82OC           ELSE
CC83OC              IF XML-TEXT = 'wssp:Password'
CC84OC                 MOVE 'Y' TO IN-ELEM PASS-XML-TAG
CC85OC              END-IF
CC86OC           END-IF
CC87OC        WHEN 'CONTENT-CHARACTERS'
CC88OC*---------------------------------------------
CC89OC* If we are in an element we are interested in,
CC90OC* then extract its value
CC91OC*---------------------------------------------
CC92OC           IF IN-ELEM = 'Y'
CC93OC              PERFORM EXTRACT-USER-ID
CC94OC           ELSE
CC95OC              IF PASS-REQ = 'Y'
CC96OC                 PERFORM EXTRACT-PASSWORD
CC97OC              END-IF
CC98OC           END-IF
CC99OC        WHEN 'END-OF-ELEMENT'
C100OC           CONTINUE
C101OC        WHEN 'START-OF-DOCUMENT'
C102OC           CONTINUE
C103OC        WHEN 'END-OF-DOCUMENT'
C104OC           CONTINUE
C105OC        WHEN 'VERSION-INFORMATION'
C106OC           CONTINUE
C107OC        WHEN 'ENCODING-DECLARATION'
C108OC           CONTINUE
C109OC        WHEN 'STANDALONE-DECLARATION'
C110OC           CONTINUE
C111OC        WHEN 'ATTRIBUTE-NAME'
C112OC           CONTINUE
C113OC        WHEN 'ATTRIBUTE-CHARACTERS'
C114OC           CONTINUE
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000553

```
           WHEN 'ATTRIBUTE-CHARACTER'
              CONTINUE
           WHEN 'START-OF-CDATA-SECTION'
              CONTINUE
           WHEN 'END-OF-CDATA-SECTION'
              CONTINUE
           WHEN 'CONTENT-CHARACTER'
              CONTINUE
           WHEN 'PROCESSING-INSTRUCTION-TARGET'
              CONTINUE
           WHEN 'PROCESSING-INSTRUCTION-DATA'
              CONTINUE
           WHEN 'COMMENT'
              CONTINUE
           WHEN 'EXCEPTION'
      *****************************************************************
      * Exception handling paragraph
      * XML code 52 means that the code page being used does not match
      * the one specified in the document. This is expected with non
      * EBCDIC data, as it has been translated to EBCDIC since we
      * received it.
      *****************************************************************
              IF XML-CODE = 52 MOVE ZERO TO XML-CODE
              ELSE
      ***************************************
      * An exception occurred during parsing
      ***************************************
                MOVE 'Y' TO PARSE-ERROR
              END-IF
           WHEN OTHER
              MOVE 'Y' TO PARSE-ERROR
        END-EVALUATE.
    XML-HANDLER-END. EXIT.
    XML01-SUBROUTINES SECTION.
      *****************************************************************
      * The parser returns everything between the <xmltag> and
      * </xmltag> tags, including all whitespace (SP, CR, LF, NL, HT)
      * These are removed before saving the xmltag content.
      *****************************************************************
    REMOVE-USELESS.
        MOVE ZERO TO I J.
        MOVE LENGTH OF XML-TEXT TO L.
        IF L > LENGTH OF WORKAREA
              MOVE LENGTH OF WORKAREA TO L.
        MOVE XML-TEXT TO WORKAREA(1 : L).
        INSPECT WORKAREA(1 : L) CONVERTING X'0D2505' TO SPACES.
        INSPECT WORKAREA(1 : L) TALLYING I FOR LEADING SPACES.
        SUBTRACT I FROM L.
        INSPECT WORKAREA(I + 1 : L) TALLYING J FOR ALL SPACES.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000554

```
C1640C     SUBTRACT J FROM L.
C1650C     IF L > LENGTH OF CA-USERID
C1660C           MOVE LENGTH OF CA-USERID TO L.
C1670C     MOVE WORKAREA(I + 1 : L) TO CA-USERID(1 : L)
C1680C     MOVE 'N' to CA-F.FM.
C1690C
C1700C EXTRACT-PASSWORD.
C1710C     MOVE ZERO TO L J.
C1720C     MOVE LENGTH OF XML-TEXT TO L.
C1730C     IF L > LENGTH OF WORKAREA
C1740C           MOVE LENGTH OF WORKAREA TO L.
C1750C     MOVE XML-TEXT TO WORKAREA(1 : L).
C1760C     INSPECT WORKAREA(1 : L) CONVERTING X'0E0F1505' TO SPACES.
C1770C     INSPECT WORKAREA(1 : L) TALLYING I FOR LEADING SPACES.
C1780C     SUBTRACT I FROM L.
C1790C     INSPECT WORKAREA(I + 1 : L) TALLYING J FOR ALL SPACES.
C1800C     SUBTRACT J FROM L.
C1810C     IF L > LENGTH OF CA-PASSWORD
C1820C           MOVE LENGTH OF CA-PASSWORD TO L.
C1830C     MOVE WORKAREA(I + 1 : L) TO CA-PASSWORD(1 : L).
C1840C     MOVE 'N' to CA-REF-REQ.
C1850C
C1860C*****************
C1870C* Return to caller
C1880C*****************
C1890C RETURN-RESPONSE.
C1900C     IF RESP-ERROR    *
C1910C*******************************************
C1920C*       Bad news, got an XML parser exception
C1930C*******************************************
C1940C        MOVE 'EXCEPTION' to CA-RSP
C1950C     ELSE
C1960C       IF USERNAME-XML-TAG-FOUND = 'N'
C1970C*************************************************
C1980C*          Ignore, it was not for us...
C1990C*          Bad news, did not find the expected XML tag
C2000C*************************************************
C2010C           MOVE 'NOT FOUND' to CA-RSP
C2020C       END IF
C2030C     END IF.
C2040C
C2050C*********************************************
C2060C* Initialise and Validate the COMMAREA
C2070C*********************************************
C2080C INIT-AND-VALIDATE.
C2090C*******************
C2100C*    Acceptable size ?
C2110C*******************
C2120C     IF EIBCALEN > 150
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000555

```
C2130C     THEN
C2140C***********
C2150C*      yes
C2160C***********
C2170C        MOVE DFHCOMMAREA-CN(1:EIBCALEN) to XML-DOCUMENT
C2180C        MOVE SPACES TO CA-USER-ID, CA-PASSWORD
C2190C     ELSE
C2200C**********
C2210C*      no
C2220C**********
C2230C        INITIALIZE DFHCOMMAREA(1:EIBCALEN)
C2240C        MOVE 'DAD COMMAREA LENGTH !' to DFHCOMMAREA
C2250C        EXEC CICS RETURN END-EXEC
C2260C     END-IF.
```

A.5 Sample header processing program - CIWSSECR

The CIWSSECR program in Example A-7 is used to insert a <wsse:Security> header into a SOAP message.

It determines, by obtaining the data from DFHFJNCTION, if it has been invoked in a SEND REQUEST phase. It also determines if it has been invoked for the correct destination, from the data in DFHWS-URI. It then obtains the user ID that the transaction is currently executing under and inserts this into the WS-Security header. This header is then written into the DFHHEADER container and control is returned to the calling program.

Example: A-7 Sample header processing program - CIWSSECR

```
CCC10C*****************************************************************
CCC20C IDENTIFICATION DIVISION.
CCC30C PROGRAM-ID.  CIWSSECR
CCC40C*****************************************************************
CCC50C*                                                               *
CCC60C* This program:-                                                *
CCC70C* 1  Checks for SEND-REQUEST function, else returns             *
CCC80C* 2  Checks for correct URI from DFHWS-URI                       *
CCC90C* 3. Obtains user ID from DFHWS-USERID container                *
CCD00C* 4. Puts completed WS-Security header into the                 *
CCD10C*    DFHHEADER container                                        *
CCD20C*****************************************************************
CCD30C
CCD40C AUTHOR.      CHERREGA  CL.
CCD50C DATE-COMPILED.
CCD60C ENVIRONMENT DIVISION.
CCD70C CONFIGURATION SECTION
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000556

```
CC180C SPECIAL-NAMES.
CC190C DATA DIVISION.
CC200C WORKING-STORAGE SECTION.
CC210C 01 WS-START.
CC220C   05 FILLER                  PIC X(44)
CC230C       VALUE 'DFHEISTG STARTS HERE ***'.
CC240C   05 HW                      PIC X(10) VALUE 'CONVERSE '.
CC250C   05 WS-RESP                 PIC S9(8) COMP-5 SYNC.
CC260C   05 WS-RESP2                PIC S9(8) COMP-5 SYNC.
CC270C   05 WS-HEAD-PTR             USAGE IS POINTER.
CC280C   05 WS-HEAD-LEN             PIC S9(8) COMP-4.
CC290C   05 WS-URI-PTR              USAGE IS POINTER.
CC300C   05 WS-URI-LEN              PIC S9(8) COMP-4
CC310C   05 WS-FUNC-LEN             PIC S9(8) COMP-4 VALUE 16.
CC320C   05 WS-PIPE-LEN             PIC S9(8) COMP-4 VALUE 9.
CC330C   05 WS-GETPATH-PTR          USAGE IS POINTER.
CC340C   05 WS-GETPATH-LEN          PIC S9(8) COMP-4.
CC350C   05 WS-FAULT-STRING         PIC X(48) value spaces.
CC360C   05 WS-FAULT-CODE           PIC S9(8) COMP-5.
CC370C   05 WS-SOAP-LEVEL           PIC S9(8) COMP-5.
CC380C     88 WS-SOAP-11  VALUE 1.
CC390C     88 WS-SOAP-12  VALUE 2.
CC400C     88 WS-NOT-SOAP VALUE 10.
CC410C   05 WS-FUNC-AREA            PIC X(16).
CC420C   05 WS-PIPE-AREA            PIC X(8).
CC430C   05 WS-XBUF-AUTH            PIC X(48)
CC440C              VALUE  'Not authorized to place order.'.
CC450C   05 WS-AUTH-FAIL-RC         PIC X(48)
CC460C              VALUE  'Authorization failed for order request.'.
CC470C   05 WS-WSSE-HEADER.
CC480C      05 WS-WSSE-01 PIC X(15) VALUE '<wsse:Security '.
CC490C      05 WS-WSSE-02 PIC X(26) VALUE 'soap-env:mustUnderstand="1"'.
CC500C      05 WS-WSSE-03 PIC X(25) VALUE ' xmlns:wsse="http://docs.o',
CC510C      05 WS-WSSE-04 PIC X(25) VALUE 'asis-open.org/wss/2004/01/',
CC520C      05 WS-WSSE-05 PIC X(75) VALUE 'oasis-200401-wss-wssecurit',
CC530C      05 WS-WSSE-06 PIC X(18) VALUE 'y-secext-1.0.xsd">'.
CC540C      05 WS-WSSE-07 PIC X(20) VALUE '<wsse:UsernameToken>'.
CC550C      05 WS-WSSE-08 PIC X(16) VALUE '<wsse:Username>'.
CC560C      05 WS-WSSE-09 PIC X(38) VALUE 'WEBUSZ    .
CC570C      05 WS-WSSE-10 PIC X(16) VALUE '</wsse:Username>'.
CC580C      05 WS-WSSE-11 PIC X(21) VALUE '</wsse:UsernameToken>'.
CC590C      05 WS-WSSE-12 PIC X(16) VALUE '</wsse:Security>'.
CC600C
CC610C 01 MYPARSER-COM-LEN           PIC S9(4) COMP-4.
CC620C 01 WS-PSW.
CC630C      05 WS-WSSE-A PIC X(33)
CC640C                        VALUE '<wsse:Password Type 'http://do'.
CC650C      05 WS-WSSE-B PIC X(33)
CC660C                        VALUE 'cs.oasis-open.org/wss/2004/01/'.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000557

```
006700      05 WS-WSSC-C PIC X(30)
006800                    VALUE 'wasis-2004Cl-wsa-username-token'.
006900      05 WS-WSSE-C PIC X(30)
007000                    VALUE 'n-profile-1 C#PasswordText'.
007100      05 WS-WSSF-P PIC X(8) VALUE 'PREDICKS '.
007200      05 WS-WSSF-P PIC X(16)  VALUE 'e/wsse:Password>'.
007300 01 DA-PARSER-WSP.
007400    03 DA USER LC             PIC X(0).
007500    03 DA PASSWORD            PIC X(0).
007600
007700 LINKAGE SECTION.
007800 01 WS-HEAD-AREA.
007900    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-HEAD-LEN.
008000
008100 01 WS-URI-AREA.
008200    02 FILLER    PIC X OCCURS 256 DEPENDING ON WS-URI-LEN.
008300
008400 01 WS-GETMAIN-AREA.
008500    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS GETMAIN LEN.
008600
008700*****************************************************************
008800* Main line code begins                                         *
008900*****************************************************************
009000 PROCEDURE DIVISION.
009100 MAIN-PROCESSING SECTION.
009200
009300     PERFORM GET-FILE.ONE.
009400
009500     PERFORM GET D-P-URL.
009600
009700     PERFORM GET-DFHWS-UR1.
009800
009900     EVALUATE WS-URI-AREA(1:22)
010000       WHEN '/ws/Quote?destination'
010100         IF WS-FUNC-AREA = 'SEND-REQUEST '
010200            PERFORM GET-USERID
010300            PERFORM GET-SOAP-HEADER
010400            PERFORM PUT SOAP HEADER
010500         END IF
010600       WHEN OTHER
010700         CONTINUE
010800     END-EVALUATE.
010900
011000     EXEC CICS RETURN END-EXEC.
011100
011200 MAIN-PROCESSING-END. EXIT.
011300
011400*****************************************************************
011500*     ***************************************     *
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000558

```
C11600*              SUBROUTINES FOLLOW                 *
C11700*     ************************************     *
C11800*************************************************
C11900
C12000*************************************************************
C12100* Retrieve the PIPELINE name from DFHWS-PIPELINE container
C12200*************************************************************
C12300 GET-PIPELINE.
C12400     EXEC CICS
C12500        GET CONTAINER('DFHWS-PIPELINE')
C12600        INTO(WS-PIPE-AREA)
C12700        FLENGTH(WS-PIPE-LEN)
C12800        RESP(WS-RESP)
C12900     END-EXEC.
C13000
C13100
C13200*************************************************************
C13300* Retrieve the function type from DFHFUNCTION container
C13400*************************************************************
C13500 GET-DFHFUNCTION.
C13600     EXEC CICS
C13700        GET CONTAINER('DFHFUNCTION')
C13800        INTO(WS-FUNC-AREA)
C13900        FLENGTH(WS-FUNC-LEN)
C14000        RESP(WS-RESP)
C14100     END-EXEC.
C14200
C14300*************************************************
C14400* Check for correct length. Create SOAP fault if
C14500* not correct. If for incoming request then we
C14600* do the work, else we exit immediately.
C14700*************************************************
C14800     IF WS-FUNC-LEN NOT = 15
C14900        MOVE WS-PATH-FAILED TO WS-FAULT-STRING
C15000        PERFORM FAULT-MESSAGE
C15100     END-IF.
C15200
C15300*************************************************************
C15400* Retrieve the URI from the DFHWS URI container
C15500*************************************************************
C15600 GET-DFHWS-URI.
C15700     EXEC CICS
C15800        GET CONTAINER('DFHWS-URI')
C15900        SET(WS-URI-PTR)
C16000        FLENGTH(WS-URI-LEN)
C16100        RESP(WS-RESP)
C16200     END-EXEC.
C16300
C16400*******************************************
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000559

```
C1630C* Copy the input container to our storage
C1640C*****************************************
C1650C     IF WS-URI-LEN > 0
C1660C        SET ADDRESS OF WS-URI-AREA TO WS-URI-PTR
C1690C        MOVE WS-URI-LEN TO WS-GETMAIN-LEN
C1700C
C1710C        EXEC CICS GETMAIN
C1720C            SET(WS-GETMAIN-PTR)
C1730C            FLENGTH(WS-GETMAIN-LEN)
C1740C        END-EXEC
C1750C
C1760C        SET ADDRESS OF WS-GETMAIN-AREA TO WS-GETMAIN-PTR
C1770C        MOVE WS-URI-AREA TO WS-GETMAIN-AREA
C1780C        SET WS-URI-PTR TO WS-GETMAIN-PTR
C1790C        SET ADDRESS OF WS-URI-AREA TO WS-URI-PTR
C1800C     ELSE
C1810C        MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C1820C        PERFORM FAULT-MESSAGE
C1830C     END-IF.
C1840C
C1850C*****************************************
C1860C* Retrieve the <wsse:Security> header
C1870C*****************************************
C1880C GET-SOAP-HEADER.
C1890C     EXEC CICS
C1900C        GET CONTAINER('DFHHEADER')
C1910C        SET(WS-HEAD-PTR)
C1920C        FLENGTH(WS-HEAD-LEN)
C1930C        RESP(WS-RESP)
C1940C     END-EXEC.
C1950C
C1960C**********************
C1970C* Retrieve the user ID
C1980C**********************
C1990C GET-USERID.
C2000C     EXEC CICS
C2010C        GET CONTAINER('DFHWS-USERID')
C2020C        INTO(WS-WSSE-09)
C2030C        FLENGTH(LENGTH OF WS-WSSE-09)
C2040C        RESP(WS-RESP)
C2050C     END-EXEC.
C2060C
C2070C*********************************
C2080C* Write the <wsse:Security> header
C2090C*********************************
C2100C PUT-SOAP-HEADER.
C2110C     EXEC CICS
C2120C        PUT CONTAINER('DFHHEADER')
C2130C        FROM(WS-WSSE-HEADER)
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000560

```
C21400        FLENGTH(LENGTH OF WS-WSSE-HEADER)
C21500        RESP(WS-RESP)
C21600     END-EXEC.
C21700     DISPLAY MM '================================'
C21800     DISPLAY MM 'Container Name: : DFHHEADER        '.
C21900     DISPLAY MM 'Content length: '    LENGTH OF WS-WSSE-HEADER.
C22000     DISPLAY MM 'Container contents: ' WS-WSSE-HEADER.
C22100     DISPLAY MM '                                   '.
C22200**
C22300
C22400*****************************************************************
C22500* The user ID has been verified. We will now set the contents of
C22600* DFHWS-USERID container with this value.This will cause the
C22700* business logic (back-end program/s) to be executed with this
C22800* user ID, using it's access rights.
C22900*****************************************************************
C23000 SET-USER-ID.
C23100     EXEC CICS
C23200         PUT CONTAINER( 'DFHWS-USERID'
C23300         FROM(CA-USER-ID)
C23400         FLENGTH(length of CA-USER-ID)
C23500         DATATYPE(DFHVALUE(CHAR))
C23600         RESP(WS-RESP)
C23700     END-EXEC.
C23800
C23900*****************************************************************
C24000* We detected that the ca_request_id field contains an invalid
C24100* request. This is a CICS-NI error.
C24200*****************************************************************
C24300 FAULT-MESSAGE-SECTION.
C24400************************
C24500* Generate a SOAP Fault
C24600************************
C24700     EXEC CICS
C24800         GET CONTAINER( 'DFHWS-SOAPLEVEL'
C24900         INTO(WS-SOAP-LEVEL)
C25000         FLENGTH(WS-HEAD-LEN)
C25100         RESP(WS-RESP)
C25200     END-EXEC.
C25300
C25400*************************************************
C25500* Perform the XML PARSE and verify the user ID
C25600*************************************************
C25700
C25800 PARSE-AND-VERIFY.
C25900*****************************************************************
C26000*  The SOAP body XML data can now be parsed by MYPARSER
C26100*    LINK to the XML parser, it will return the CA-USER-ID
C26200*                                          and CA PASSWORD
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000561

```
C2630C*                                          or   EXCEPTION'
C2640C*                                          or   NOT FOUND'
C2650C*                                          or   BAD COMMA'
C2660C*****************************************************************
C2670C     EXEC CICS  LINK PROGRAM('MYPARSER')
C2680C                    COMMAREA(WS-HEAD-AREA)
C2690C                    LENGTH(MYPARSER-COPLEN)
C2700C     END EXEC.
C2710C     IF WS HEAD AREA(1:9) = 'EXCEPTION' or
C2720C                            'NOT FOUND' or
C2730C                            'BAD COMMA'
C2740C*****************************************************
C2750C*  Error found during the XML PARSE program execution
C2760C*****************************************************
C2770C         MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C2780C         PERFORM FAULT-MESSAGE
C2790C     ELSE
C2800C         MOVE WS-HEAD-AREA to CA-PARSER-RSP
C2810C************************************
C2820C*  OK, data found, so do the VERIFY
C2830C************************************
C2840C         EXEC CICS VERIFY
C2850C            PASSWORD(CA-P455WORD)
C2860C            USERID(CA-USER-ID)
C2870C            RESP(WS-RESP)
C2880C         END-EXEC
C2890C*****************
C2900C*  Successful ?
C2910C***************
C2920C         IF WS RES- = DFHRESP(NORMAL)
C2930C            PERFORM SET-USER-ID
C2940C         ELSE
C2950C            MOVE WS-NOT-AUTH TO WS-FAULT-STRING
C2960C            PERFORM FAULT-MESSAGE
C2970C         END-IF
C2980C     END-IF.
C2990C*****************************************
C3000C* MOVE CORRECT VERSION OF FAULTCODE
C3010C*****************************************
C3020C     IF WS SOAP = 1 MOVE DFHVALUE(CLIENT) TO WS FAULT CODE
C3030C     ELSE
C3040C        MOVE DFHVALUE(SENDER) TO WS-FAULT-CODE
C3050C     END-IF
C3060C
C3070C     EXEC CICS SOAPFAULT CREATE
C3080C               FAULTSTRING(WS-FAULT-STRING)
C3090C               FAULTSTRLEN(LENGTH OF WS-FAULT-STRING)
C3100C               FAULTCODE(WS-FAULT-CODE)
C3110C     END EXEC.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000562

```
CC120C FAULT-MESSAGE-END. EXIT.
```

A.6 Sample header processing program - CIWSSECS

Program CIWSSECS in Example A-8 is similar to CIWSSECH (see Appendix A.2, "Sample header processing program - CIWSSECH" on page 505) except that no password verification is done.

Example. A-8 Sample header processing program CIWSSECS

```
CC010C  PROCESS CICS
CC020C IDENTIFICATION DIVISION.
CC030C PROGRAM-ID.  CIWSSECS
CC040C*****************************************************************
CC050C*                                                               *
CC060C* This program:                                                 *
CC070C* 1. Checks for RECEIVE REQUEST function, else returns         *
CC080C* 2. Obtains data from DFHHEADER container                      *
CC090C* 3  Uses the PYPARSER program to extract the CA-USER-ID       *
CC100C*    field from the XML message contained w i t h DFHHEADER     *
CC110C* 4  Puts CA-USER-ID  into DFHWS-USERID conta ner              *
CC120C*****************************************************************
CC130C
CC140C AUTHOR.      Grant Ward Ahlr.
CC150C DATE COMPILED.
CC200C ENVIRONMENT DIVISION.
CC210C CONFIGURATION SECTION.
CC220C SPECIAL-NAMES.
CC230C DATA DIVISION.
CC240C WORKING-STORAGE SECTION
CC250C 01 WS-START.
CC260C   05 FILLER                    PIC X(44)
CC270C       VALUE 'CIWSSECS WORKING STORAGE STARTS HERE '.
CC280C   05 WS-RESP                    PIC S9(8) COMP-5 SYNC.
CC290C   05 WS-RESP2                   PIC S9(8) COMP-5 SYNC.
CC300C   05 WS HEAD PTR                USAGE IS POINTER.
CC310C   05 WS-HEAD-LEN                PIC S9(8) COMP-4.
CC320C   05 WS-URI-PTR                 USAGE IS POINTER.
CC330C   05 WS-URI-LEN                 PIC S9(8) COMP-4.
CC340C   05 WS-FUNC-LEN                PIC S9(8) COMP-4 VALUE 16.
CC350C   05 WS-GETPATH-PTR             USAGE IS POINTER
CC360C   05 WS-GETPATH- FN             PIC S9(8) COMP-4.
CC370C   05 WS-FAULT-STRING            PIC X(48) va ue spaces.
CC380C   05 WS-FAULT-CODE              PIC S9(8) COMP-4.
CC390C   05 WS-SOAP-LEVEL              PIC S9(8) COMP-4.
CC400C     88 WS SOAP 11  VALUE 1.
CC410C     88 WS SOAP 12  VALUE 2.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000563

```
CC42OC     88 WS-NOT-SCAP VALUE 10.
CC43OC   03 WS-FUNC-AREA             PIC X(16).
CC44OC   03 WS-NOT-AUTH              PIC X(40)
CC45OC               VALUE  'Not authorized to place order.'.
CC46OC   05 WS-AUTH-PAT PI           PIC X(40)
CC47OC               VALUE  'Authorization failed for order request.'.
CC48OC
CC49OC 01 MYPARSER COM-LEN            PIC S9(4) COMP 5.
CC5OOC 01 DA PARSER RSP.
CC51OC    03 DA USER ID             PIC X(8).
CC52OC    03 DA-PASSWORD            PIC X(8).
CC53OC
CC54OC LINKAGE SECTION.
CC55OC 01 WS-HEAD-AREA.
CC56OC    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-HEAD-LEN.
CC57OC
CC58OC 01 WS-URI-AREA.
CC59OC    02 FILLER    PIC X OCCURS 256 DEPENDING ON WS-URI-LEN.
CC6OOC
CC61OC 01 WS-GETMAIN-AREA.
CC62OC    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-GETMAIN-LEN.
CC63OC
CC64OC*****************************************************************
CC65OC* Main line code beg *                                           *
CC66OC*****************************************************************
CC67OC PROCEDURE DIVISION.
CC68OC MAIN-PROCESSING SECTION.
CC69OC
CC7OOC**********************************
CC71OC*  Receive the SOA- Body namespace
CC72OC**********************************
CC73OC     PERFORM GET-DFHFUNCTION.
CC74OC
CC75OC**********************************
CC76OC*  Receive the SOAP Body namespace
CC77OC**********************************
CC78OC     PERFORM GET-DFHWS-URI.
CC79OC     IF WS URI AREA(1:4) NOT = 'wrq:'
CC8OOC        EXEC CICS RETURN END EXEC
CC81OC     END IF.
CC82OC
CC83OC**********************************
CC84OC*  Receive the SOAP Body namespace
CC85OC**********************************
CC86OC     PERFORM GET-SOAP-HEADER.
CC87OC
CC88OC*****************************************************************
CC89OC*  The SOAP body XML data can now be parsed by MYPARSER
CC9OOC*    LINK to the XML parser, it will return the DA USER ID
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000564

```
CC530C*                                          or   EXCEPTION'
CC540C*                                          or   NOT FOUND'
CC550C*                                          or   BAD COMMA'
CC560C*****************************************************************
CC570C     MOVE SPACES TO CA-PARSER-RSP
CC570C     EXEC CICS  link PROGRAM('MYPARSER')
CC580C                    COMMAREA(WS-HEAD-AREA)
CC590C                    LENGTH(MYPARSER_COPLEN)
C10000     END EXEC.
C10100     IF WS HEAD AREA(1:9) = 'EXCEPTION' or
C10200                           'NOT FOUND' or
C10300                           'BAD COMMA'
C10400*****************************************************
C10500*  Error found during the XML PARSE program execution
C10600*****************************************************
C10700         MOVE WS-ALTE-FAILED TO WS-FAULT-STRING
C10800         PERFORM FAULT-MESSAGE
C10900     ELSE
C11100*************************************
C11200*  OK, data found, so set the user ID
C11300*************************************
C11400         MOVE WS-HEAD-AREA to CA-PARSER-RSP
C12300         PERFORM SET-USER-ID
C12800     END-IF.
C12900*************************
C13000*  and it is the end, bye bye
C13100*************************
C13700     EXEC CICS RETURN END-EXEC.
C13800
C13900 MAIN PROCESSING END. EXIT.
C14000
C14100*****************************************************
C14200*     ************************************     *
C14300*             SUBROUTINES FOLLOW               *
C14400*     ************************************     *
C14500*****************************************************
C14600
C14700*************************************************************
C14800* Retrieve the function type from UPPERFUNCTION container
C14900*************************************************************
C15000 GET-DFHFUNCTION.
C15100     EXEC CICS
C15200        GET CONTAINER( DFHFUNCTION')
C15300        INTO(WS-FUNC-AREA)
C15400        FLENGTH(WS-FUNC-LEN)
C15500        RESP(WS-RESP)
C15600     END-EXEC.
C15700
C15800*****************************************************
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000565

```
C18400* Check for correct length. Create SOAP fault if
C18500* not correct. If for incoming request then we
C18600* do the work, else we exit immediately.
C18700*************************************************
C18800     IF WS-FUNC- Eh NOT - 15
C18900        MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C19000        PERFORM FAULT-MESSAGE
C19100     ELSE
C19200        IF WS FLAG AREA NOT = 'RECEIVE REQUEST'
C19300           EXEC CICS xE JPY END EXEC
C19400        END-IF
C19500     END-IF.
C19600
C19700*************************************************************
C19800* Retrieve the URI from the DFHWS-URI container
C19900*************************************************************
C20000 GET-DFHWS-URI.
C20100     EXEC CICS
C20200        GET CONTAINER( DFHWS JRI')
C20300        SET(WS-URI-PTR)
C20400        FLENGTH(WS-URI-LEN)
C20500        RESP(WS-RESP)
C20600     END-EXEC.
C20700
C20800*****************************************
C20900* Copy the input container to our storage
C21000*****************,***********************
C21100     IF WS-URI-.Eh > 0
C21200        SET ADDRESS Of WS JRI AREA IC WS URI PTr
C21300        MOVE WS URI LEN TO WS GETMAIN LEN
C21400
C21500        EXEC CICS GETMAIN
C21600            SET(WS-GETMAIN-PTR)
C21700            FLENGTH(WS-GETMAIN-LEN)
C21800        END-EXEC
C21900
C22000        SET ADDRESS Of WS-GETMAIN-AREA IC WS-GETMAIN-PTR
C22100        MOVE WS URI AREA TO WS GETMAIN AREA
C22200        SET WS URI PTr  O WS GETMAIN PTR
C22300        SET ADDRESS Of WS JRI AREA IC WS URI PTr
C22400     ELSE
C22500        MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C22600        PERFORM FAULT-MESSAGE
C22700        EXEC CICS RETURN END-EXEC
C22800     END-IF.
C22900
C23000*****************,***********************
C23100* Retrieve the <wsse:Security> header
C23200*************************************
```


Document Id: D3N4ZF3 4827 11E7 B66D A6E29A16B6C
Confidentiality: IBM



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000566

```
C20300 GET-SOAP-HEADER.
C20400     EXEC CICS
C20500        GET CONTAINER( DFHHEADER')
C20600        SET(WS-HEAD-PTR)
C20700        FLENGTH(WS-HEAD-LEN)
C20800        RESP(WS-RESP)
C20900     END-EXEC.
C21000
C21100*-------------------------------------
C21200* Copy the input container to our storage
C21300*-------------------------------------
C21400     IF WS-HEAD-LEN > 0
C21500        SET ADDRESS OF WS-HEAD-AREA TO WS-HEAD-PTR
C21600        MOVE WS-HEAD-LEN TO WS-GETMAIN-LEN OF PARSER-CONLEN
C21700
C21800        EXEC CICS GETMAIN
C21900            SET(WS-GETMAIN-PTR)
C22000            FLENGTH(WS-GETMAIN-LEN)
C22100        END EXEC
C22200
C22300        SET ADDRESS OF WS-GETMAIN-AREA TO WS-GETMAIN-PTR
C22400        MOVE WS-HEAD-AREA TO WS-GETMAIN-AREA
C22500        SET WS-HEAD-PTR TO WS-GETMAIN-PTR
C22600        SET ADDRESS OF WS-HEAD-AREA TO WS-HEAD-PTR
C22700     ELSE
C22800        MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C22900        PERFORM FAULT-MESSAGE
C23000        EXEC CICS RETURN END-EXEC
C23100     END-IF.
C23200
C23300*-------------------------------------------------------------
C23400* The user ID has been verified, we will now set the contents of
C23500* DFHWS-USERID container with this value.This will cause the
C23600* business logic (back-end program(s)) to be executed with this
C23700* user ID, using it's access rights.
C23800*-------------------------------------------------------------
C23900 SET-USER-ID.
C24000     EXEC CICS
C24100         PUT CONTAINER( DFHWS-USERID')
C24200         FROM(CA-USER-ID)
C24300         FLENGTH(length of CA-USER-ID)
C24400         DATATYPE(DFHVALUE(CHAR))
C24500         RESP(WS-RESP)
C24600     END-EXEC.
C24700
C24800*-------------------------------------------------------------
C24900* We determine that the ca_request_id field contains an invalid
C25000* request. This is a CLIENT error.
C25100*-------------------------------------------------------------
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000567

```
C2520C FAULT-MESSAGE SECTION.
C2530C**********************
C2540C* Generate a SOAP Fault
C2550C**********************
C2560C     EXEC CICS
C2570C          GET CONTAINER( DFHWS-SOAPLEVEL )
C2580C          INTO(WS-SOAP-LEVEL)
C2590C          FLENGTH(WS-READ-LEN)
C2600C          RESP(WS-RESP)
C2610C     END-EXEC.
C2620C
C2630C*************************************
C2640C* MOVE CORRECT VERSION OF FAULTCODE
C2650C*************************************
C2660C     IF WS-SOAP-11 MOVE DFHWSJF(CLIENT) TO WS-FAULT-CODE
C2670C     ELSE
C2680C        MOVE DFHWSJF(SENDER) TO WS-FAULT-CODE
C2690C     END-IF
C2700C
C2710C     EXEC CICS SOAPFAULT CREATE
C2720C               FAULTSTRING(WS-FAULT-STRING)
C2730C               FAULTSTRLEN(LENGTH OF WS-FAULT-STRING)
C2740C               FAULTCODE(WS-FAULT-CODE)
C2750C     END-EXEC.
C2760C FAULT-MESSAGE-END. EXIT.
```

A.7 Sample header processing program - WSATHND

The WSATHND C program shown in Example A-9 is used to monitor the exchange of registration and protocol service messages between WebSphere Application Server and CICS. It writes messages to the CESO transient data queue.

Example A-9 Sample header processing program - WSATHND

```
/*****************************************************************/
/*                                                               */
/* This program will perform the following functions when used as */
/* a  message handler in either the DFHWSATP pipeline or the      */
/* DFHWSATR pipeline:                                            */
/*                                                               */
/* 1) If MESSAGES ON is set to 1, it will write the following    */
/*    message:                                                   */
/*        WSAT: REACHED HANDLER   function                       */
/*    where function is the contents of the C-H-FUNCTION container */
/*    (RECEIVE REQUEST,SEND RESPONSE,SEND REQUEST,RECEIVE RESPONSE, */
/*    PROCESS-REQUEST,PC-RESPONSE,or HANDLER-CARRY). Recall that */
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000568

```
/*    When the function is NO-RESPONSE, then the handler is being   */
/*    invoked after processing a request, when there is no response */
/*    to be processed.                                              */
/*                                                                  */
/* 2) If the function is not NO-RESPONSE,then:                      */
/*    a) If FULL_MESSAGES_ON is set to 1, it will write the         */
/*       following message:                                         */
/*          MSG1: contents of the DFHREQUEST container              */
/*                                                                  */
/*    b) If MESSAGES_ON is set to 1, it will write the following    */
/*       message:                                                   */
/*          MSG1: ACTION: action                                    */
/*       where action is the character string aaaaaa which comes    */
/*       at the end of the following character strings found in the */
/*       DFHREQUEST container:                                      */
/*          http://schemas.xmlsoap.org/ws/2004/10/wscoor/aaaaaa     */
/*       and has one of the following values: RegisterResponse,     */
/*       Prepared,Committed, Register, Prepare, Commit, ReadOnly,   */
/*       Aborted, Abort, Rollback                                   */
/*                                                                  */
/* The messages are written to the CESO transient data queue which  */
/* is normally an extrapartition transient data queue with DDname   */
/* CESOUT                                                           */
/*                                                                  */
/********************************************************************/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

/*******************************************************************/
/* The following flags can be set to either 0 or 1.                */
/*******************************************************************/
#define MESSAGES_ON        1
#define FULL_MESSAGES_ON   1

void check_response(char* containerr);

int main(char* argv[])
{
   /*****************/
   /*** VARIABLES ***/
   /*****************/

   /* Pointer to COMMAREA */
   char* commptr;

   /* Container names */
   char* cont_function = "DFHFUNCTION    \0";
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000569

```
char* cont_request  = "DFHREQUEST      \0";

/* Contents and length of DFHFUNCTION container */
char* function;
long int functionLen;

/* Contents and length of DFHREQUEST container */
char* message;
long int messageLen;

/* log message */
char* entryLogMsg;

char action[] = "               \0";

char* state1 =
     "http://schemas.xmlsoap.org/ws/2004/10/wscoor/RegisterResponse";
char* state2 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepared";
char* state3 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Committed";
char* state4 = "http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register";
char* state5 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepare";
char* state6 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Commit";
char* state7 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/ReadOnly";
char* state8 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Aborted";
char* state9 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Abort";
char* state10= "http://schemas.xmlsoap.org/ws/2004/10/wsat/Rollback";

/***************/
/*** PROGRAM ***/
/***************/

EXEC CICS ADDRESS EIB(dfheiptr) CWAREA(commptr);

/*****************************************************************/
/* Store contents of DFHFUNCTION container in function variable */
/*****************************************************************/
EXEC CICS GET CONTAINER(cont_function) NODATA
          FLENGTH(functionLen);
check_response("WSATHND.getContainer.dfhfunction",

function = (char*) malloc(1 sizeof(char) * (1 + functionLen));

EXEC CICS GET CONTAINER(cont_function)
          INTO(function) FLENGTH(functionLen);
check_response("WSATHND.getContainer.dfhfunction");

*(function + functionLen)   '\0';

/*****************************************************/
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000570

```
/* Display message: WSAT: REACHED HANDLER - function  */
/*****************************************************/

entryLogMsg = (char*) malloc(sizeof(char*) *
                             (strlen("WSAT: REACHED HANDLER - ") +
                              strlen(function) + 1
                             )
                            );
strcpy(entryLogMsg, "WSAT - REACHED HANDLER   ");
strcat(entryLogMsg, function);
if (MESSAGES_ON == 1)
  printf("%s\n", entryLogMsg);

free(entryLogMsg);

/*****************************************************************/
/* Store contents of DFHREQUEST container in message variable */
/*****************************************************************/

if (!(strncmp(function, "NO-RESPONSE     ", 16) == 0))
{
    EXEC CICS GET CONTAINER(cont_request) NODATA
                 FLENGTH(messageLen);
    check_response("WSATHND.get.requestLen");

    message = (char*) malloc(sizeof(char*) * (messageLen+1));

    EXEC CICS GET CONTAINER(cont_request)
                 INTO(message) FLENGTH(messageLen),
    check_response("WSATHND.get.request");

    *(message+messageLen) = '\0';

    if (FULL_MESSAGES_ON == 1)
    {
      printf("WSAT:%s:\n", message);
    }

    /****************/
    /* ACTION STATE */
    /****************/


    if (strstr(message, state1) != NULL)
        strcpy(action, "RegisterResponse\0");
    else if (strstr(message, state2) != NULL)
        strcpy(action, "Prepared       \0");
    else if (strstr(message, state3) != NULL)
        strcpy(action, "Committed      \0");
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000571

```
        else if (strstr(message, state4) != NULL)
            strcpy(action, "Register        \0");
        else if (strstr(message, state5) != NULL)
            strcpy(action, "Prepare         \0");
        else if (strstr(message, state6) != NULL)
            strcpy(action, "Commit          \0");
        else if (strstr(message, state7) != NULL)
            strcpy(action, "ReadOnly        \0");
        else if (strstr(message, state8) != NULL)
            strcpy(action, "Aborted         \0");
        else if (strstr(message, state9) != NULL)
            strcpy(action, "Abort           \0");
        else if (strstr(message, state10) != NULL)
            strcpy(action, "Rollback        \0");

        /* (MESSAGES_ON == 1)
        printf("MSAT: ACTION=%s.\n", action);

        free(message);
    }

    EXEC CICS RETURN;
    check_response("mSATHND.rel,rm");

    return 0;
}


/* Check response of CICS commands - ensure no errors occur */
void check_response(char* identifier)
{
     if ((dfheiptr->eibresp + dfheiptr->eibresp2) != 0)
     {
         char* errStart = "ERROR: ";
         char* errEnd   = ". See stdout for responses";
         char* fullMsg;

         if (dfheiptr->eibresp == DFHRESP(NOTAUTH))
            printf("NOTAUTH Response found");
         if (dfheiptr->eibresp == DFHRESP(CCSIDERR))
            printf("CCSIDERR Response found");
         if (dfheiptr->eibresp == DFHRESP(CHANNELERR))
            printf("CHANNELERR Response found");
         if (dfheiptr->eibresp == DFHRESP(CONTAINERR))
            printf("CONTAINERR Response found");
         if (dfheiptr->eibresp == DFHRESP(INVREQ))
            printf("INVREQ Response found");
         if (dfheiptr->eibresp    DFHRESP(LENGERR))
            printf("LENGERR Response found");
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000572

```
        printf("Error: %s\n", identifier);
        printf("Error resp1: %d\n", dfheipt->eibresp);
        printf("Error resp2: %d\n", dfheipt->eibresp2);

        fullMsg = (char*) malloc(strlen(errStart) +
                                 strlen(identifier) +
                                 strlen(errEnd) + 1) *
                                 sizeof(char*));

        strcpy(fullMsg, errStart);
        strcat(fullMsg, identifier);
        strcat(fullMsg, errEnd);

        EXEC CICS SEND TEXT FROM(fullMsg)
                    LENGTH(strlen(fullMsg));

        free(fullMsg);

        exit;
    }
|
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000573

B

How the DES, AES, and HMAC algorithms work

Some readers may want or need a more detailed explanation of one or more of the following:

- How the DES algorithm works
- How the AES algorithm works
- How the HMAC algorithm of FIPS PUB 198 works

In this appendix we attempt to provide an overview of this information that does not require an understanding of advanced mathematics.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000574

B.1 How DES works

DES makes some use of permutations. A permutation of a set of elements is an arrangement of the elements of the set in some order. For example, there are six permutations of the set {1,2,3}: 123, 132, 213, 231, 312, and 321. The permutation 123 and the permutation 132 are considered to be different permutations because the order of the elements is different.

A sketch of the DES enciphering computation is given in Figure B-1. The 64 bits of the input block to be enciphered are first subjected to a permutation. In DES the permuted input has bit 58 of the input as its first bit, bit 50 of the input as its second bit, and so on as specified in FIPS Pub 46-3 with bit 7 of the input as its last bit. Figure B-1 shows the leftmost 32 bits of the permuted input as L_0, and the rightmost 32 bits of the permuted input as R_0. The permuted input block is then the input to a complex key-dependent computation described in "The cipher function f" on page 547. The output of that computation, called the preoutput, is then subjected to a permutation that is the inverse of the initial permutation.

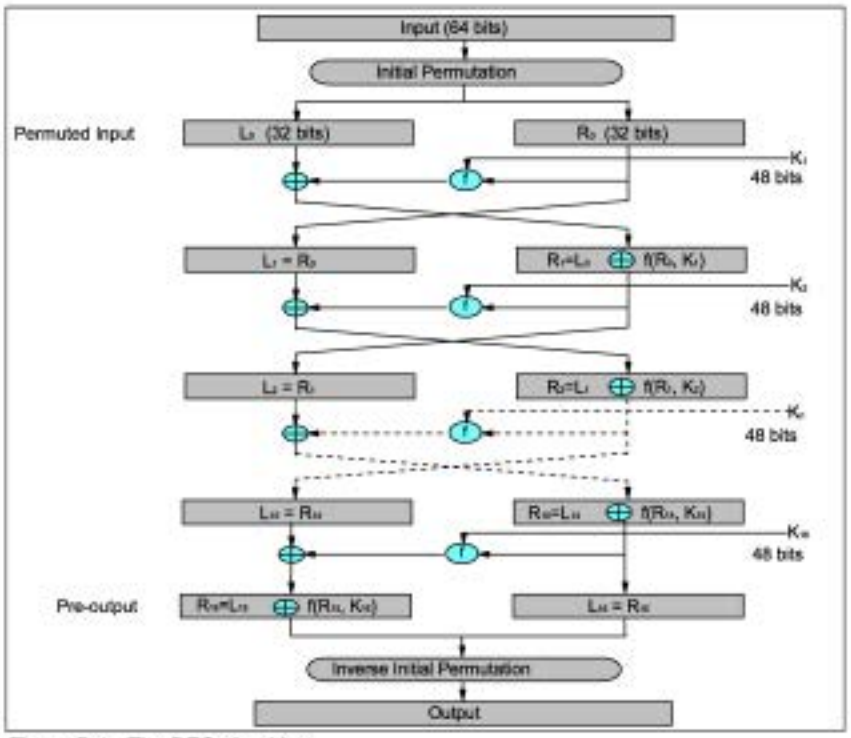


Figure B-1 The DES algorithm

Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineNotary.net
Page 575/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000575

The computation that uses the permuted input block as its input to produce the preoutput block consists, but for a final interchange of blocks, of 16 iterations of the calculation:

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$$

R_{n-1} represents the rightmost 32 bits of the output from the previous iteration. K_n represents a permuted selection of 48 bits chosen from the 64-bit key by the key schedule algorithm described in "Key schedule algorithm" on page 548. Thus the cipher function f operates on two blocks, one of 32 bits and one of 48 bits; it produces a block of 32 bits, which is then XORed with L_{n-1} to produce R_n. (XOR is a bit-by-bit exclusive OR operation in which $1 \oplus 1 = 0, 1 \oplus 0 = 1, 0 \oplus 1 = 1, 0 \oplus 0 = 0$. XOR is equivalent to a bit-by-bit addition modulo 2.) The preoutput block is then $R_{16}L_{16}$.

The cipher function f

A sketch of the calculation of f(R,K) is given in Figure B-2.

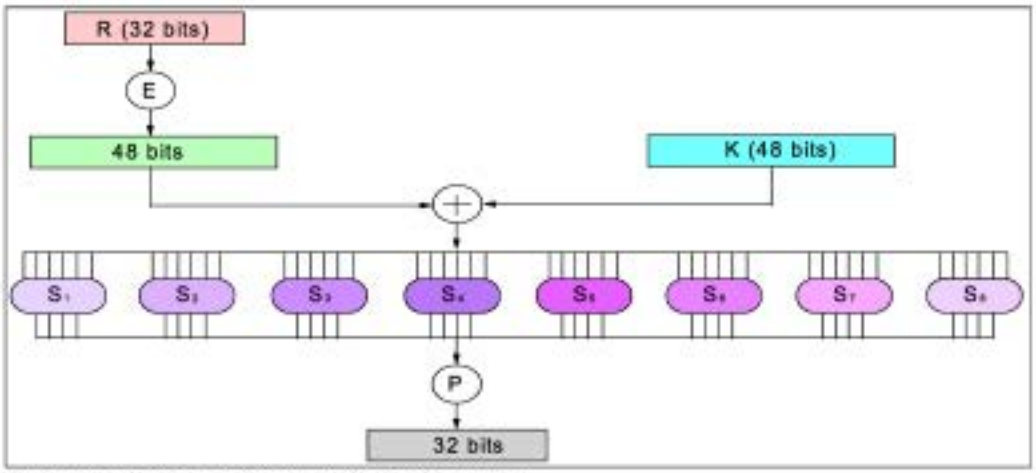


Figure B-2 The cipher function f of the DES algorithm

E denotes a function that takes a block of 32 bits as input and produces a block of 48 bits as output. The first three bits of E(R) are the bits in positions 32, 1, and 2 of R, the last 2 bits of E(R) are the bits in position 32 and 1 of R, and the remaining bits of E(R) are chosen from the bits of R according to a table specified in FIPS Pub 46-3.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 576/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000576

Each of the unique selection functions $S_1, S_2 \ldots S_8$ takes a 6-bit block of input and yields a 4-bit block as output as specified in FIPS Pub 46-3.

The 8 blocks of 4 bits each are consolidated into a single block of 32 bits, which forms the input to the permutation function P. P yields a 32-bit output from a 32-bit input by permuting the bits of the input block, again as specified in FIPS Pub 46-3. The output is then the output of the cipher function f for the inputs R and K.

Key schedule algorithm

Figure B-3 shows a sketch of the key schedule calculation of the DES algorithm. The bits of the key are numbered 1 through 64. The bits of C_0 are respectively bits 57,49, 41, 33, 25 . .. 44 and 36 of the key. The bits of D_0 are respectively bits 63, 55, 47, 39, 31, . . 12, and 4 of the key. Permuted Choice 1 does not select any of the eight parity bits of the key for either C_0 or D_0.

C_1 and D_1 are obtained from C_0 and D_0 respectively by one left shift. C_2 and D_2 are obtained from C_1 and D_1 respectively by one left shift. C_8 and D_8 are obtained from C_7 and D_7 by one left shift, and C_{16} and D_{16} are obtained from C_{15} and D_{15} by one left shift. In all other cases C_n and D_n are obtained from C_{n-1} and D_{n-1} by two left shifts. In all cases, by *one left shift* is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3, ... 28, 1.

Permuted Choice 2 selects the 48 bits of K_n from the 56 bits of C_nD_n as specified by a table in FIPS Pub 46-3. The first bit of K_n is the 14th bit of C_nD_n, the second bit is the 17th, and so on with the 47th bit of K_n being the 29th bit of C_nD_n, and the 48th bit of K_n being the 32nd bit of C_nD_n.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000577

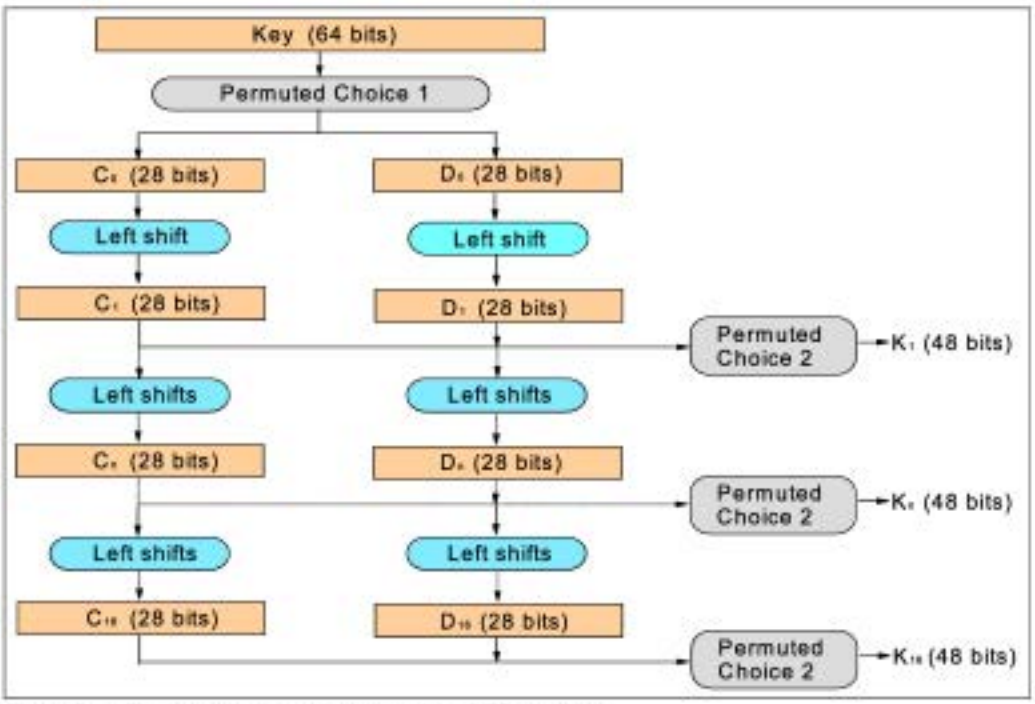


Figure B-3 The key schedule calculation of the DES algorithm

B.2 How AES works

The AES algorithm requires an initial set of 4 words of key data, and each of the Nr rounds requires 4 words of key data. Thus the algorithm requires 4 + 4*Nr = 4(Nr+1) words of key data. Consequently the algorithm specifies a Key Expansion routine that accepts the cipher key as input and uses it to generate a key schedule containing 4(Nr+1) words as output. The first Nk words of the key schedule are the first Nk words of the cipher key. For example, if we specify the following 128-bit cipher key:

2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

then the Key Expansion routine generates a key schedule containing 4(10 + 1) = 44 words, of which the first 4 words are as follows:

w_0 = 2b7e1516 w_1 = 28aed2a6 w_2 = abf71588 w_3 = 09cf4f3c

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 578/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000578

Example B-1 shows pseudo code that describes how AES encrypts data. The pseudo code uses four variables.

- The variable *in* is an array of 16 bytes that contains the 128 bits to be encrypted.
- The variable *out* is an array of 16 bytes that contains the data after it has been encrypted.
- The variable *w* is an array of 4*(Nr + 1) words: it contains the key schedule that the Key Expansion routine generated.
- The variable *state* is a rectangular array having 4 rows and 4 columns of bytes that represent intermediate encryption results.

Example B-1 Pseudo code for AES encryption

```
Cipher( byte in[16], byte out[16], word w[4*(Nr+1)])
begin
    byte state[4,4]

    state = in

    AddRoundKey(state, w[0,3] )

    for round = 1 step 1 to Nr-1
       SubBytes(state)
       ShiftRows(state)
       MixColumns(state)
       AddRoundKey(state, w[4*round, 4*round+3])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[4*Nr, 4*Nr+3])

    out = state
end
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000579

The input array, *in*, is copied into the *state* array according to the scheme shown in Example B-2

Example B-2 state = in

```
s[0,0] = in[0]  s[0,1] = in[4]  s[0,2] = in[8]  s[0,3] = in[12]
s[1,0] = in[1]  s[1,1] = in[5]  s[1,2] = in[9]  s[1,3] = in[13]
s[2,0] = in[2]  s[2,1] = in[6]  s[2,2] = in[10] s[2,3] = in[14]
s[3,0] = in[3]  s[3,1] = in[7]  s[3,2] = in[11] s[3,3] = in[15]
```

For example, if the data to be encrypted is:

32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

then the initial value of the *state* array would be as shown in Table B-1.

Table B-1 state = in

32	**88**	**31**	**e0**
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

Note that the first column of the *state* array contains the first 32-bit word of the input, the second column of the *state* array contains the second word of the input, and so forth. The *state* can hence be interpreted as a one-dimensional array of 32-bit words (columns): w0, w1, w2, w3.

The encryption operations are then conducted on this *state* array. After an initial Round Key addition, the *state* array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first Nr-1 rounds because it does not include the MixColumns() transformation. The final state is then copied to the output.

The individual transformations - SubBytes(), ShiftRows(), MixColumns(), and AddRoundKey() - which make up the round function that processes the state array are described in the following sections.

SubBytes() transformation

The SubBytes() transformation is a substitution that operates independently on each byte of the *state* array using the substitution table set forth in Figure 7 of FIPS Pub 197. The substitution table contains 16 rows and 16 columns. If, for example, the $s_{1,1}$ element of the *state* array contained the byte 53, then the substitution value would be found at the intersection of the row with index '5' and

Document Id: [illegible]
Confidentiality: [illegible]
Page 000580

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000580

the column with index '3' in the substitution table. This would result in 53 being replaced with ed.

The creators of AES used complex mathematics to create the substitution table in such a way that it would have an inverse substitution that could be used in a decryption operation.

ShiftRows() transformation

In the ShiftRows() transformation the bytes in the second row of the *state* array are cyclically shifted left by one column. The bytes in the third row are cyclically shifted left by two columns. The bytes in the last row are cyclically shifted left by three columns. The first row is not shifted. See Figure B-4.



Figure B-4 ShiftRows() cyclically shifts the last three rows in the state array

MixColumns() transformation

The MixColumns() transformation, shown in Figure B-5, replaces the four bytes in column c of the state array, namely $\{s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}\}$, with four new bytes $\{s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}\}$ whose values are computed according to the following equations:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

In these equations $a \oplus b$ represents the XOR operation, which is equivalent to addition modulo 2, and $x \bullet y$ represents the multiplication of two elements in the finite field GF(2^8). It is beyond the scope of this book to explain the mathematics of finite fields. However, we will note that the creators of AES chose the coefficients in the preceding equations to have certain specific values in the finite

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 581/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000581

field GF(2^8) so that the MixColumns() transformation would have an inverse transformation that could be used in a decryption operation.

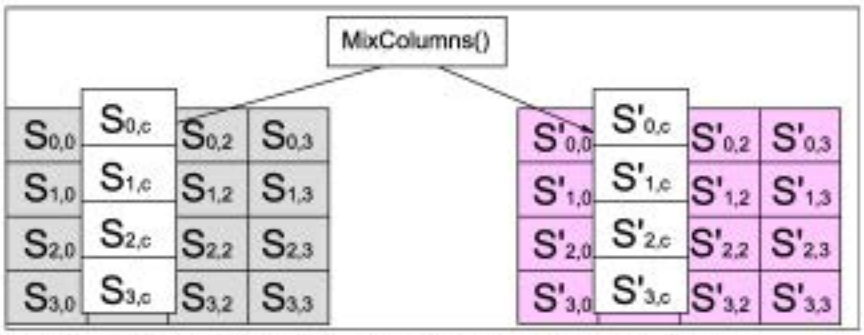


Figure B-5 MixColumns() operates on the state array column-by-column

AddRoundKey() transformation

Figure B-6 shows that the AddRoundKey() transformation XORs each column of the *state* array with a word from the key schedule. In Figure B-6 I is equal to four times the number of the round. The AddRoundKey() transformation performs an XOR operation between the first column of the state array and word w_i from the key schedule in order to create the first column of the new *state* array. Likewise, it performs an XOR operation between the second column of the *state* array and word w_{i+1} from the key schedule to create the second column of the new state array. Likewise, column three is XORed with w_{i+2}, and column four is XORed with w_{i+3}.

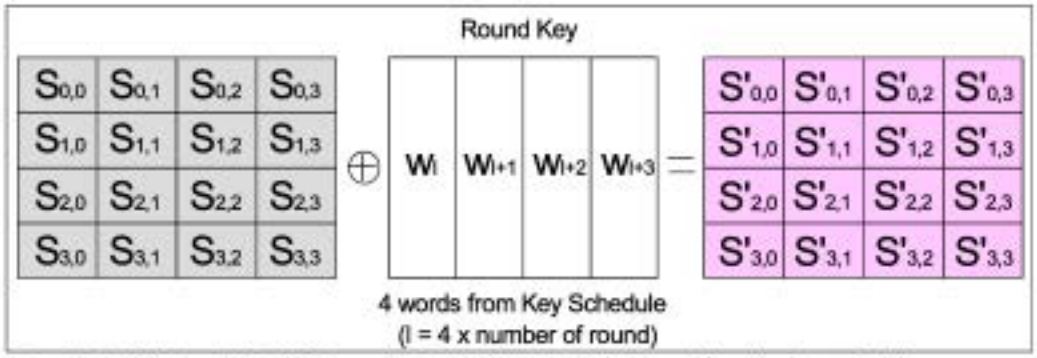


Figure B-6 AddRoundKey() XORs each column of the state with a word from the key schedule

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 582/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000582

B.3 How the HMAC algorithm of FIPS PUB 198 works

FIPS PUB 198 describes an HMAC using the parameters shown in Table B-2.

Table B-2 Parameters for the HMAC algorithm

Parameter	Description
B	Block size (in bytes) of the input to the hash function
H	Hash function (such as SHA-1)
ipad	Inner pad, the byte x'36' repeated B times
K	Secret key shared between the originator and the intended receiver(s)
K_0	The key K after any necessary pre-processing to form a key of length B
L	Block size (in bytes) of the output of the hash function
opad	Outer pad, the byte x'5C' repeated B times
l	The number of bytes of MAC
text	The data on which the HMAC is calculated; *text* does not include the padded key. The length of text is n bits

A well-known practice with MACs is to truncate their output; that is, the length of the MAC used is less than the length L of the output of the hash function. When a truncated HMAC is used, the l leftmost bytes of the HMAC computation are used as the MAC. The output length, l, should be no less than four bytes, so $4 \le l \le L$.

To compute a MAC over the data *text* using the HMAC function, FIPS PUB 198 specifies the performance of the following operation:

$$MAC(text, K) = HMAC(K, text) = H((K_0 \oplus opad) \| H((K_0 \oplus ipad) \| text))$$

where $a \oplus b$ indicates an Exclusive-OR operation and $\|$ indicates a concatenation operation. Table B-3 describes the step-by-step process in the HMAC algorithm.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000583

Table B-3 The HMAC algorithm

Step	Description of step
1	If the length of $K = B$: set $K_0 = K$. Go to step 4.
2	If the length of $K > B$: hash K to obtain a string L bytes long, then append $(B-L)$ zeros to create string K_0 which is B bytes long That is, $K_0 = H(K) \| 00...00$. Then go to step 4.
3	If the length of $K < B$: append zeros to the end of K to create a string K_0 which is B bytes long. (For example, if K is 20 bytes in length and $B = 64$, then K will be appended with 44 zero bytes X'00').
4	Exclusive-OR K_0 with *ipad* to produce the string $K_0 \oplus ipad$. This string has length B.
5	Append the stream of data 'text' to the string resulting from step 4: $(K_0 \oplus ipad) \| text$
6	Apply the hash function H to the stream generated in step 5: $H((K_0 \oplus ipad) \| text)$
7	Exclusive-OR K_0 with *opad* to produce the string $K_0 \oplus opad$. This string has length B.
8	Append the result from step 6 to step 7: $(K_0 \oplus opad) \| H((K_0 \oplus ipad) \| text)$
9	Apply the hash function H to the result from step 8: $H((K_0 \oplus opad) \| H((K_0 \oplus ipad) \| text))$
10	Select the leftmost I bytes of the result of step 9 as the MAC

The successful verification of a MAC does not completely guarantee that the accompanying message is authentic: there is a chance that a source with no knowledge of the key can present a purported MAC on the plaintext message that will pass the verification procedure. For example, an arbitrary purported MAC of I bits on an arbitrary plaintext message may be successfully verified with an expected probability of $(1/2)^I$. Therefore, in general, if the MAC is truncated, then its length I should be chosen as large as is practical, with at least half as many bits as the output block size, L.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000584

Document Id: [illegible]
Client Security: [illegible]



Page 556(5)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000585

Abbreviations and acronyms

2PC	Two-phase commit
AAT	Application Assembly Tool
AOR	Application-Owning Region
API	Application Programming Interface
APPC	Advanced Program-to-Program Communications
ASCII	American Standard Code for Information Interchange
B2B	Business-to-business
CCI	Common Client Interface
CEX2	Crypto Express2
CICS	Customer Information Control System
CICS TG	CICS Transaction Gateway
CORBA	Common Object Request Broker Architecture
CPACF	CP Assist for Cryptographic Functions
DPL	Distributed Program Link
DSA	Digital Signature Algorithm
EAR	Enterprise Application Archive
EBCDIC	Extended Binary Coded Decimal Interchange Code
ECI	External Call Interface
EIS	Enterprise Information Systems
EJB	Enterprise JavaBeans™
EPI	External Presentation Interface
ESI	External Security Interface
ESM	External Security Manager
EXCI	External CICS Interface
FTP	File Transfer Protocol
GTF	Generalized Trace Facility
GUI	Graphical user interface
HFS	Hierarchical File System
HTML	Hypertext Transfer Protocol
HTTP	Hypertext Markup Language
IBM	International Business Machines
ICSF	Integrated Cryptographic Service Facility
IDE	Integrated development environment
IRC	inter-region communication
ISC	Inter-System Communication
ITSO	International Technical Support Organization
J2C	J2EE Connector Architecture (also known as JCA)
J2EE	Java 2 Enterprise Edition
JAAS	Java Authentication and Authorization Service
JAR	Java Archive
JAX-RPC	Java API for XML-based RPC
JCA	J2EE Connector Architecture (also known as J2C)
JDBC	Java Database Connectivity
JDK™	Java Developer's Kit
JKS	Java Keystore
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface™
JNI	Java Native Interface
JSP	JavaServer Page

© Copyright IBM Corp. 2006. All rights reserved.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000586

JSSE	Java Secure Sockets Extension
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
LPAR	Logical Partition
LTPA	Lightweight Third-Party Authentication
LUW	Logical Unit of Work
MRO	Multi Region Operation
PEM	Password Expiration Management
PKDS	Private Key Data Set
QoS	Quality of service
RAD	Rational Application Developer
RAR	Resource Adapter Archive
RPC	Remote procedure call
RRMS	Recoverable Resource Management Services
RRS	Resource Recovery Services
RSA	Rivest, Shamir, and Adelman (encryption)
SAF	System Authorization Facility
SAX	Simple API for XML
SDK	Software Development Kit
SNA	Systems Network Architecture
SOAP	Simple Object Access Protocol (also known as Service-Oriented Architecture Protocol)
SQL	Structured query language
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
TPV	Tivoli® Performance Viewer
UDDI	Universal Description, Discovery and Integration
UOW	Unit of Work
URI	Uniform resource identifier
URL	Uniform resource locator
URN	Uniform resource name
VSAM	Virtual Storage Access Method
WMQ	WebSphere MQ
WS-A	Web Services - Addressing
WS-AT	Web Services - Atomic Transaction
WS-BA	Web Services - Business Activity
WS-C	Web Services - Coordination
WSDL	Web Services Description Language
WSGW	Web Services Gateway
WS-I	Web Services Interoperability
WS-Security	Web Services - Security
XMI	XML metadata interchange
XML	Extensible Markup Language
XSD	XML Schema Definition

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000587

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 560. Note that some of the documents referenced here may be available in softcopy only.

- *Application Development for CICS Web Services*, SG24-7126
- *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461
- *Web Services Handbook for WebSphere Application Server 6.1*, SG247257
- *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303
- *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864
- *Websphere MQ Security in an Enterprise Environment*, SG24-6814

Other publications

These publications are also relevant as further information sources:

- *Enterprise COBOL for z/OS V3R3 Programming Guide*, SC27-1412
- *Enterprise COBOL for z/OS V3R3 Language Reference*, SC27-1408
- *CICS Web Services Guide V3.1*, SC34-6458
- *CICS TS V3.1 RACF Security Guide*, SC34-6249
- *CICS TS V3.1 Application Programming Reference*, SC34-6434
- ICSF Overview, SA22-7519
- RACF Command Reference, SA22-7687
- *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454
- *Your Web Services Atomic Transaction operations: Beginner's guide to classic transactions, data recovery, and mapping to WS-Atomic Transactions*. IBM developerWorks (September 2, 2004)

© Copyright IBM Corp. 2006. All rights reserved.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000588

- *WebSphere MQ for z/OS System Setup Guide V6.0*, SC34-6583
- *WebSphere MQ - Transport for SOAP*, SC34-6651
- *Websphere MQ Security*, SC34- 6588

Online resources

These Web sites and URLs are also relevant as further information sources:

- CICS library

  `http://www.ibm.com/software/htp/cics/library/`

- WebSphere Application Server library

  `http://www.ibm.com/software/webservers/appserv/was/library/index.html`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000589

Index

A
aborted 413
AddRoundKey 553
Advanced Encryption Standard (AES) 150
AES algorithm
  how it works 545
alert protocol 197
APARs
  PK10849 439
  PK16509 437
  PK20393 113
application sample 76
Application Server Toolkit V6.1 (AST) 305
asymmetric cryptography 150
ATM
  application 385
atomic transaction 381
AtomicClient 439–440, 445, 461
auditing 150
authentication 152, 231
  basic 302
authorization 152
Automatic Teller Machine (ATM) 382
availability 126
  configuring for 101

B
basic 217
basic authentication 217, 302
  sample 292, 308
bean-managed transaction 450
block cipher 154
bus
  see service integration bus

C
C language 34
C++ 34
catalog application 122, 439, 461
catalog manager 146
catalog manager application 67
  configuration transaction ECFG 97, 122, 465
  testing 90, 100, 124, 145
CatalogEar 97
CatalogAtomic.ear 450
CatalogSer.ear 280
CatalogSec_WS-Security.ear 265, 346, 370
CatalogSec_WS-Security_HTTPS.ear 267
CEDA 119, 443
CEMT 120
certificate 274, 319
certificate revocation 184
certification paths 182
channels 59
CICS 295
  certificate 322
  COBOL programs 65
  configuring 311
  encryptor 221
  Execution Diagnostic Facility (EDF) 125
  security 214
  service provider 57
    preparation 36
    using HTTP 76–77
    using WMQ 116
  service requester 41
    local optimization 45
    preparation 42
    timeout considerations 96
    using HTTP 94–95
    using WMQ 121
  trace 105
  Web services assistant 34, 62
    DFHLS2WS 63
    DFHWS2LS 64
  Web services support 33
    catalog manager application 67
    development tools 61
    high availability 101, 126
    overview 34
    problem determination 104
    resources 47
    resources checklist 55
  WS-Security Component 279
CICS default user ID 215
CICS region user ID 215

© Copyright IBM Corp. 2006. All rights reserved.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000590

CICSplex 102
CICS-to-CICS 112
cipher 154
  suite 193
Cipher Block Chaining (CBC) 155
cipher suite 290
CIWSMSGH 80, 93, 118, 500
CIWSSECH 265–269, 505
CIWSSECR 267, 527
CIWSSECS 269, 303, 534
COBOL 34
code page 74, 77
code page conversion 77
commit 393, 413
commitand 413
Communications Manager 102
compilation process 410
confidentiality 152
connection 79
connector 439
container-managed transaction 450
containers 59
  context container 59
  control container 59
  header container 60
  user container 61
CoordinationContext 400
  elements 468
    Address 469
    CoordinationType 469
    Expires 468
    Identifier 468
    ReferenceProperties 469
    RegistrationService 469
  example 467
  sample created by CICS 402
  sample created by WebSphere 403
coordinator 406, 436
CORBA 8
core standards 8
CP Assist for Cryptographic Functions (CPACF) 209, 321
CPIH 41, 93
CPIL 116
credentials 311
Crypto Express2 (CEX2) feature 209, 321
cryptographic hardware 209
cryptography 151, 153
  hardware 321
  public key 160
  secret key 153
  symmetric 154
cryptosystem 151

D
daisy chain 436
Data 155
Data Encryption Algorithm (DEA) 155
Data Encryption Standard (DES) 156
DB2 tasks 439
decrypting 153
default user ID, CICS 215
DES algorithm
  how it works 545
development tools 61
DFHSRING 275, 299
DFHLS2WS 34, 63
DFHPIHS 429
DFHPISNT, message handler module 429, 440
DFHRESPONSE container 82
DFHRSLRI, URIMAP resource 426
DFHWS2LS 34, 64
DFHWSATH, header processing program 420, 427–429, 440
DFHWSATP, provider pipeline 421–422
DFHWSATR, requester pipeline 421
  PIPELINE definition 424
DFHWSATX, message handler 421, 424
Diffie-Hellman 160
digital signature 153, 168, 330
Digital Signature Algorithm (DSA) 171
dispatchOrder.ear 96
DispatchOrderAtomic.ear 402
distributed routing 102–103
DPL subset 87
dynamic program routing 102, 104
dynamic response 47
dynamic routing 104

E
EAR file 99
eavesdropping 215
ECFG 97, 122, 465
Electronic Codebook (ECB) 155
electronic data interchange (EDI) 4
Elliptic Curve Digital Signature Algorithm (ECDSA) 173

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000591

encrypting 153
encryption algorithm 195
endpoint
  listener 131
  reference 391
enterprise bean 450
error 330
  handling 16
error handling 197
ExampleAppClient.ear 87
ExampleAppDispatchOrder 88
ExampleAppDispatchOrder.ear 89, 462
Execution Diagnostic Facility (EDF) 125
Extensible Markup Language
  see XML

F
FTP 9, 276

G
gateway
  instance 131
    configuring 140
  service 131, 135
    configuring 139, 141
    invoking 147
    publishing 145
    testing 145

H
handler program 499
handshake protocol 193, 198
hardware 209
  cryptography 321
hash algorithm 193
hashing 153, 164
heuristic decision 416
HFS file system 74
high availability 101, 126
  with HTTP 101
  with WMQ 126
HMAC algorithm
  how it works 545
HTTP 7, 36, 73, 215
hub configuration 426

I
IBM Java SDK 277
ibm-webservices-bnd.xmi 259, 304
ibm-webservicesclient-bnd.xmi 259, 304
ibm-webservicesclient-ext.xmi 259, 304
ibm-webservices-ext.xmi 259, 304
identification 152
identity assertion 244
iKeyMan 277
IMS 65
inbound service 131
in-doubt window 415
Integrated Cryptographic Service Facility (ICSF)
210
integrity 152
intermediary 242
intermediate gateway
  identity assertion 244
Internet 3
iterated block cipher 151

J
J2EE
  application 146
  connector 439
J2EE Connector Architecture (JCA) 496
Java SDK 277
Java Transaction API (JTA) 432
JAX-RPC
  handler 130, 131
JCA 439
JDBC 439
JDBC provider 459
JMS 9, 439
JVM 69

K
key agreement protocol 189
key exchange algorithm 193
key exchange protocol 189
key pair
  generation 319
keyring 274
  creating 274
key schedule algorithm 548
keystore 277

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000592

L
local optimization 45

M
master secret 201
mediator 130–131
medium 222
message authentication codes (MAC) 166
message handler 35, 90, 116, 296
  introduction 57
  SOAP 57
  writing 90
message handler program 300
message information headers 394
mixcolumns 552
mustUnderstand 14, 265, 311, 316, 351, 375
MVS 102
MYPARSER 268, 522

N
namespace 12
Namespaces 12
National Institute of Standards and Technology (NIST) 155
non repudiation 153
non terminal message 415
notification message 414

O
one way message 414
outbound service 131

P
Parallel Sysplex 102
participant 417, 436
performance 304
PIPELINE 49
  attributes
    CONFIGFILE 50
    SHELF 50
    WSDIR 49
  defining 83, 95, 119
pipeline 35, 49, 60, 118, 265, 301
  configuration file 51
    customizing 90
pipeline alias transaction 48, 57, 93
PLT 34
Port definition 29
prepared 413
presumed abort 383
presumed trust 244
problem 316
  determination 104
protocol transformation 132
public key cryptography 150
  cryptography
    public key 151

R
RACF 325
  keyring 299
Rational Application Developer 265, 280
readonly 413
recoverable 383
Redbooks Web site 560
  Contact us xxii
reference
  parameter 393
  property 393
Register
  request 404
  response 406
remote procedure call (RPC) 17
replay 413
request consumer 240
request generator 240, 359
RequestStream 431
response consumer 241, 338
response generator 241
rollback 383, 413
RPC 17
  style 33
RSA algorithm 167

S
sample application 76, 287, 409
scalability 101
secret 152
secret key cryptography 153
  cryptography
    secret key 151
Secure Sockets Layer (SSL) 215
security
  auditing 153
  authentication 152

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000593

authorization 152
confidentiality 152
exposures 215
HTTP 217
identification 152
integrity 152
non repudiation 153
prefixing 300
setup 299
SIT parameters 300
SOAP message 217, 257
see WS-Security
token 302
transport-level 217
comparison with SOAP message security 241
WMQ 224, 293
testing 291
self-signed 319
service
broker 5
gateway 135
see gateway service
inbound 131
mediator 130
outbound 131
provider 5
requestor 5
service integration bus 129–130, 139
configuration 132
securing CICS Web services 242
service provider 76
service provider mode 303
service requester
installing 87
timeout considerations 124
service requester setup 304
service-oriented approach 4
service-oriented architecture 3–4
characteristics 5
shillrowe 552
signature
processing 347
SIT parameters 76, 300
DFLTUSER 215
ENCRYPTION 219
KEYRING 219, 281, 300
MAXSSLTCBS 219
SCC 251, 300
SECPRFX 251, 300
SSLCACHE 220
SSLDELAY 219
TCPIP 76
XTRAN 251, 300
XUSER 251, 300
SMTP 9
SNIFFER 136, 440, 511
sample output 519
SOA
see service-oriented architecture
SOAP 6–9
1.1 specification 9
1.2 specification 10
binding 30
body 15, 469
outbound data conversion 41
communication styles 17
document 17
RPC 17
encodings 17
literal 17
SOAP encoding 17
envelope 12–13
fault 16
fault messages 271, 314, 324
headers 14, 468
intermediary 14
introduction 12
message 296, 330
message encryption 357
message handler 35, 57
message security 217, 225, 241
messaging mode 17
MustUnderstand 15
namespaces 12
security headers 284
server 7
Simple Object Access Protocol 9
validation in CICS 50
SOAPFAULT
commands
ADD 56
CREATE 56
generating a SOAP fault in CICS 271
SOCKETCLOSE 79
spoofing 215
SSL/TLS 273
CICS support for 219

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000594

creating a key ring 271
enabling CICS support 279, 281
enabling WebSphere support 276
testing 282
with WMQ 226
standards
JAX RPC 133
JCA 456
JTA 432
SOAP 9
UDDI 10
WS-Addressing 381
WS-Atomic Transaction 381
WS-Coordination 381
WSDL 10
WS-Security 11
WS-Transactions 11
XML 8
static response 47
Station-to-Station (STS) protocol 197
strong 222
subbyles 551
subkey 154
surrogate 214
surrogate access 313
switch profiles 324
symmetric cryptography 154
Sysplex Distributor 102, 128

T
tampering 215
tbsCertificate 175
TCP/IP 417
load balancing 102
TCPIP.SIT parameter 79
TCPIPSERVICE 55, 78, 261
attributes
AUTHENTICATE 216
PORTNUMBER 79
PROTOCOL 79
SSL 220
terminal message 415
testing 90, 100, 124, 270, 282, 300, 461, 489
throughput 204
token 302
trading
CICS 105
transactional scope 365
transactions
atomic 389, 401
enabling CICS support 415
business activity 401
classic 382, 383
two phase commit 383
transport 111, 241
security 217
Transport Layer Security (TLS) 191
trust token 244
trusted 160
two phase commit (2PC) 383
presumed abort 384
walkthrough 395
with WS-AT 412

U
UDDI 8
Unicode 77
Universal Description, Discovery, and Integration (UDDI) 10
URI 123
URIMAP 47, 120, 135, 252, 302
attributes
USERID 223
defining 85, 135, 253
for serving HFS file 135
setting the userID 252
UsernameToken 303
UsernameTokenGenerator 263, 309

V
verification 172

W
Waico 382
weak 222
Web 287
Web service 53
Web Service Description Language (WSDL) 8
Web services 3, 287
assistant 62
binding file 55
definition 7
introduction 6
properties 7
Web Services Description Language

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000595

see WSD_
Web Services Gateway 130
Web Services Interoperability Organization
see WS-
Web services versus service-oriented architectures 6
Web technologies 3
WEBSERVICE 59
attributes
VALIDATION 53
WSBIND 53
WSDLFILE 53
dynamically installing 65
WebSphere 239
certificate 324
key store 326
MQ 111
service requester
deploying 86, 90–99
using HTTP 87, 98
WebSphere Application Server 298
administrative console 68
connection pooling 89
WebSphere Developer for zSeries 65
WebSphere MQ
see WMQ
WMQ 112
configuration 114
defining queues 115
trigger process 116
shared queues 127–128
WS Binding 260
WS Binding page 263, 309
WS Extension page 261, 307
WS-Addressing 381, 390
CICS Registration Service endpoint 396
endpoint reference 391
specification 390
WSATHND 444, 509
WS-Atomic Transaction (WS-AT) 36, 381, 387, 408
Completion protocol 410
daisy chain test scenario 462
testing 489
enabling CICS support 420, 421
enabling WebSphere support 432, 459
resynchronization processing 415
simple test scenario 441
testing 461
specification 400
Two-Phase Commit protocol 411
Durable 2PC 411
Volatile 2PC 411
typical scenario 439
walkthrough 387, 414
WS-Business Activity specification 400
unsupported by CICS 401
WS-Coordination 381
Activation service 390
Coordination service 398
Protocol services 396
Registration service 398
specification 397
walkthrough 406
WSDL 10, 18
binding 19, 27
bindings 30
definition 24
document 19
anatomy 19
message 19, 25
namespaces 23
operation 19, 26
port 19, 29
port type 19, 26
service 19
service definition 29
SOAP binding 30
type 19
types 24
Web Services Description Language 10
WSGW 130
WS-I
basic profile 10
introduction 6
WS-Security 226, 257
CICS support 229
configuration
files 265, 311–313, 350, 372
configuring CICS 265
example 231, 255, 311, 372
message authentication 231
specifications 226
test scenario 270, 313, 350, 372
typical scenario 228
WebSphere client configuration 260, 306
WebSphere configuration files 258, 304
with WMQ 234
WS-Security Component 229

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000596

X
X.509 230, 286
XML 7, 9
XML encryption 357
XML Services for the Enterprise (XSE) 65

Z
z/OS 102, 210
  Communications Manager 102
  Parallel Sysplex 102

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000597

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000598

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000599

Implementing CICS Web Services



Redbooks

Configuring and securing Web services in CICS Transaction Server

Connecting CICS to a service integration bus

Enabling atomic Web services

The Web services support in CICS Transaction Server Version 3.1 enables your CICS programs to be Web service providers or requesters. CICS supports a number of specifications including SOAP Version 1.1 and Version 1.2, and Web services distributed transactions (WS-Atomic Transaction).

This IBM redbook will help you configure the CICS Web services support for both HTTP and WebSphere MQ based solutions. We show how Web services can be used to integrate J2EE applications running in WebSphere Application Server with COBOL programs running in CICS.

It begins with an overview of Web services standards and the Web services support provided by CICS TS V3.1. Complete details for configuring CICS Web services using both HTTP and WebSphere MQ are provided next, along with the steps for using Web services to connect to CICS from a service integration bus. The book then shows how CICS Web services can be secured using a combination of Web Services Security (WS-Security) and transport-level security mechanisms such as SSL/TLS. Finally, it demonstrates how atomic Web services transactions can be configured to allow WebSphere and CICS resource updates to be synchronized.

This book concentrates on implementation specifics such as security, transactions, and availability. The companion redbook *Developing CICS Web Services* (SG24-7126) presents detailed information on developing CICS Web services.

SG24-7206-01 ISBN 0738494534

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000601

JURAT ATTACHMENT

A notary public or other officer completing this certificate verifies only the identity of the individual who signed the document to which this certificate is attached, and not the truthfulness, accuracy, or validity of that document.

STATE OF Texas }

COUNTY OF Tarrant }

The foregoing instrument was subscribed and sworn before me this date of 08/29/2024, by Nathaniel Frank-White

This notarial act was an online notarization.



(Notary Seal)

Notary's Signature _____

Registration No.: 133645929

Commission Expiration Date: March 15, 2026

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000602

7

Securing Web services

While you can exercise very strict control over the access to your applications from conventional 3270 terminals, your Web services clients are likely to be in remote locations, and you will be faced with new security issues.

When implementing a CICS Web services solution, you need to consider questions like the following:

- Will *authentication* be done by CICS itself or in an external server such as WebSphere Application Server?
- What *authorization* mechanisms will be used to protect access to the CICS system and access to resources such as transactions, files, and databases?
- How will you protect the *confidentiality* of data that is transported between the different tiers of the physical configuration?

In this chapter, we provide an introduction to security technologies as they relate to securing CICS Web services.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000242

7.1 Traditional CICS security

In a CICS environment, the assets you normally want to protect are the application programs and the resources that are accessed by the application programs. To prevent disclosure, destruction, or corruption of these assets, you must control access to the CICS region and to different CICS components.

You can limit the activities of a CICS user to only those functions that the user is authorized to use by implementing one or more of the following CICS security mechanisms:

- Transaction security

  This ensures that users who attempt to run a transaction are entitled to do so.

- Resource security

  This ensures that users who use CICS resources, such as files and transient data queues, are entitled to do so.

- Command security

  This ensures that users who use CICS system programming commands are entitled to do so.

- Surrogate security

  This ensures that a *surrogate* user is authorized to act on behalf of another user.

When CICS security is active, requests to attach transactions, and requests by transactions to access resources, are associated with a user ID. When a user makes such a request, CICS calls the external security manager (such as RACF) to determine if the user ID has the authority to complete the request. If the user ID does not have the correct authority, CICS denies the request.

In many cases, a user is a human operator, interacting with CICS through a terminal or a workstation. However, the user can also be a Web browser user or in a Web services solution, a program executing in a client system.

7.1.1 CICS user IDs

When a human operator signs on to a CICS region at the start of a terminal session, he or she is challenged to provide a user ID and password. The user ID remains associated with the terminal until the terminal operator signs off. Transactions executed from the terminal, and requests made by those transactions, are associated with that user ID.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000243

For connections from Web users, there are other ways that the user ID of a CICS transaction can be identified, including:

- A Web browser user is challenged to provide HTTP basic authentication information (a user ID and password). The transaction that services the client's request, and further requests made by that transaction, are associated with that user ID.
- A client program that is communicating with CICS using the Secure Sockets Layer (SSL) supplies a client certificate to identify itself. The security manager maps the certificate to a user ID. The transaction that services the client's request, and further requests made by that transaction, are associated with that user ID.

In addition to these transport-level authentication mechanisms, Web service clients can also pass authentication data in the SOAP message.

Special CICS user IDs

There are two particular user IDs that CICS uses in addition to those that identify individual end users. These are:

Region user ID The CICS region user ID is used for authorization checking when the CICS system (rather than an individual user of the system) requests access to system resources such as CICS data sets and other servers.

Default user ID When a user does not sign on, CICS assigns a default user ID to the user. It is specified in the SIT parameter DFLTUSER. In the absence of more explicit identification, it is used to identify TCP/IP clients that connect to CICS. You should give very little authority to the default user ID.

For a complete discussion of traditional CICS security, refer to *CICS TS V3.1 RACF Security Guide*, SC34-6249.

7.2 Security exposures

An end-to-end security solution addresses the security exposures found along the path of a request from an end client to a target service, including any intermediary services that route, or participate in, the service request.

To illustrate potential security exposures in a Web services environment, we use the bank teller scenario shown in Figure 7-1. The bank teller (Web service requester or client) connects over the Internet to the bank's data center where the Web service provider runs.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000244

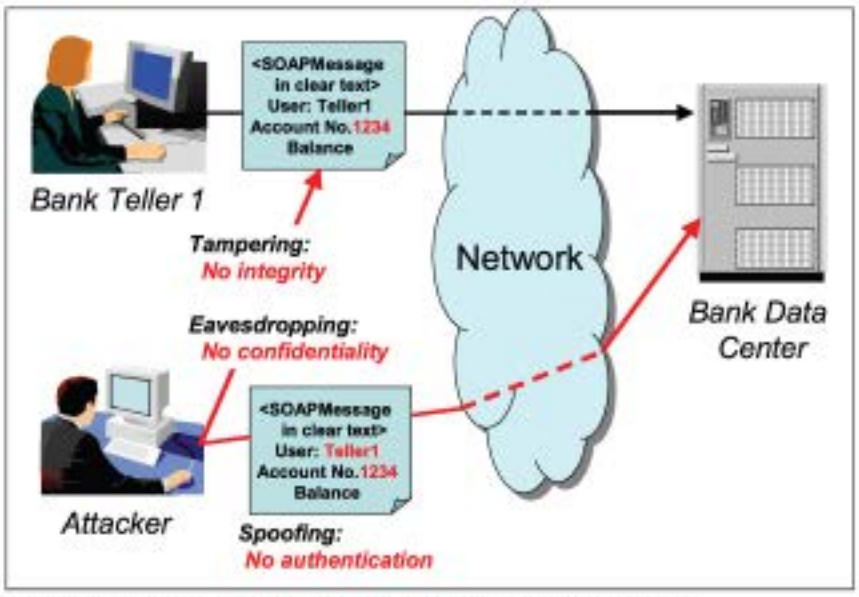


Figure 7-1 Potential security exposures in a Web services environment

If the bank has not applied any security, it has the following exposures:

- **Spoofing**

  An attacker posing as the bank teller could send a SOAP message to the service provider to get confidential information or to withdraw money from another customer's account.

  The bank can eliminate this security exposure by requiring that the bank teller authenticate herself.

- **Tampering**

  An attacker could intercept the SOAP message between the Web service requester and provider and modify the message, for example, to deposit the money into another account by changing the account number. Because there is no integrity constraint, the Web service provider does not verify whether the message has been altered and accepts the modified transaction.

  The bank can eliminate this security exposure by implementing digital signatures.

- **Eavesdropping**

  An attacker could intercept the SOAP message and read the information contained in the message because it has been sent in clear text. The attacker

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 245/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000245

could obtain confidential customer or bank information such as account numbers and balances.

The bank can eliminate this security exposure by encrypting the SOAP message.

In order to protect against the risk of these security exposures, the bank can use a combination of two types of security to secure its Web services environment:

- Transport-level security

  Transport level security mechanisms such as HTTPS can be used to secure Web services. In "Transport security" we review how different transport-level security mechanisms can be used to secure a CICS Web services solution.

- SOAP message security

  The Web Services Security model introduces a set of interrelated specifications to form a layering approach to security. When products implement these specifications, they send security information within the SOAP message itself as SOAP message headers. In "SOAP message security" on page 226 we discuss how SOAP headers can be used to secure a CICS Web services solution.

7.3 Transport security

Web services messaging relies on two protocol layers, the transport layer and the SOAP layer. Security can be implemented within either, or both, of these layers.

In this section we review how the different transport-level security mechanisms can be used to secure a CICS Web services solution. In "Comparison of transport versus SOAP message security" on page 241 we look at the main decision factors for choosing between transport-level and SOAP message security.

7.3.1 HTTP transport

When a CICS Web service is invoked using HTTP, standard HTTP security mechanisms can be used to authenticate the Web service client and to ensure message integrity and confidentiality.

Basic authentication

HTTP basic authentication is a simple challenge and response mechanism with which a server can request authentication information (a user ID and password) from a client. The client passes the authentication information to the server in an

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000246

HTTP Authorization header. The authentication information is in base-64 encoding.

The AUTHENTICATE attribute on the CICS TCPIPSERVICE resource definition specifies the authentication and identification scheme to be used for inbound TCP/IP connections for the HTTP protocol. You enable HTTP basic authentication by specifying BASIC for the AUTHENTICATE attribute.

A CICS service provider application can be protected by HTTP basic authentication. However, the HTTP basic authentication scheme can only be considered a secure means of authentication when the connection between the Web service client and the CICS region is secure. If the connection is insecure, the scheme does not provide sufficient security to prevent unauthorized users from discovering and using the authentication information for a server. If there is a possibility of a password being intercepted, basic authentication should be used in combination with SSL, so that SSL encryption is used to protect the user ID and password information.

CICS support for SSL/TLS

CICS supports both the SSL 3.0 and TLS 1.0 protocols. HTTPS connections will automatically use the TLS protocol, unless the client specifically requires SSL 3.0.

A CICS service provider application can be secured using HTTPS. HTTPS has the following advantages:

- It provides a very fast and secure transport for CICS Web services.
- It provides for authentication through either HTTP basic authentication or a client X.509 certificate.
- It provides integrity for the data passed between the service requester and CICS.
- It provides confidentiality for the data passed between the service requester and CICS by using efficient secret key cryptography.
- It can be used with hardware cryptographic devices that can significantly reduce the cost of SSL handshakes. You can customize your encryption settings to use only the cipher suites that use the Integrated Cryptographic Service Facility (ICSF).
- It is mature and similarly implemented by most vendors, and therefore, is subject to few interoperability problems.

A CICS service requester application can also use HTTPS to invoke a service provider application, with advantages similar to those listed.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000247

Enabling CICS support for SSL/TLS

To activate SSL/TLS support in a CICS TS V3.1 region, you must specify values for the following system initialization parameters:

- ENCRYPTION={STRONG | WEAK | MEDIUM}

  Specifies the cipher suites that CICS uses for secure TCP/IP connections. When a secure connection is established between the client and CICS, the most secure cipher suite supported by both is used.

  - Use ENCRYPTION=STRONG when you can tolerate the overhead of using high encryption if the other system requires it.
  - Use ENCRYPTION=WEAK when you want to use encryption keys up to 40 bits in length.
  - Use ENCRYPTION=MEDIUM when you want to use encryption keys up to 56 bits in length.

- KEYRING=*key-ring-name*

  Specifies the name of a key ring in the RACF database that contains keys and certificates used by CICS. It must be owned by the CICS region user ID. You can create an initial key ring with the DFIHSRING exec in the CICS-supplied SDFHSAMP target library.

- MAXSSLTCBS={8 | *number*}

  Specifies the maximum number of S8 TCBs that are available to CICS to process SSL connections. The S8 TCBs are created and managed in the SSL pool. An S8 TCB is only used by a task for the duration of the SSL processing.

  The maximum number of S8 TCBs that you can set is 1024. Increasing the number of available TCBs allows more simultaneous SSL connections to take place. However, increasing the number of TCBs too much will impact storage below the line.

- SSLDELAY={600 | *number*}

  Specifies the length of time in seconds for which CICS retains session IDs for secure sockets connections in the SSLCACHE. Session IDs are tokens that represent a secure connection between CICS and an SSL client. The session ID is created and exchanged between the SSL client and CICS during the SSL handshake.

  While the session ID is retained by CICS within the SSLDELAY period, CICS will re-establish an SSL connection with a client without the significant overhead of another SSL handshake. The value is a number of seconds in the range 0 through 86400. The default value is 600.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000248

Increasing the value of the SSLDELAY parameter retains the session IDs in the cache for longer, thereby optimizing the time it takes to perform SSL negotiations.

- SSLCACHE={CICS|SYSPLEX}

  Specifies whether CICS should use the local SSL cache in the CICS region, or share the cache across multiple CICS regions by using the coupling facility. If the cache is shared between a number of CICS regions, the throughput of SSL connections will improve.

  Caching across a sysplex can only take place when the regions accept SSL connections at the same IP address.

In 'Enabling SSL/TLS' on page 273 we show how we enabled our CICS region to support SSL connections from our Web service client running in WebSphere Application Server.

Activating SSL for a TCPIPSERVICE

To activate SSL support for a given connection, you must set the value of the SSL attribute on the connection's TCPIPSERVICE definition to one of the following:

- YES
  If you set the value to YES, CICS will send a server certificate to the client.
- CLIENTAUTH
  If you set the value to CLIENTAUTH, CICS will send a server certificate to the client and the client must send a client certificate to CICS.

  If you specify SSL(CLIENTAUTH) you can also specify AUTHENTICATE(CERTIFICATE) in order to map the client certificate to a RACF user ID which is then used as the identity for the client.

Cipher suites

A cipher suite must be supported by the operating system in order for CICS to use it. The list of cipher suites supported by z/OS depends on your release of the operating system. Table 7-1 and the example that follows it are for z/OS 1.4 and CICS TS V3.1. If you have a higher release of z/OS, you may have additional cipher suites available.

Table 7-1 Cipher suites

Cipher suite	Encryption algorithm	Key length	MAC algorithm
01	No encryption		MD5
02	No encryption		SHA-1

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000249

Cipher suite	Encryption algorithm	Key length	MAC algorithm
03	RC4	40 bits	MD5
04	RC4	128 bits	MD5
05	RC4	128 bits	SHA-1
06	RC2	40 bits	MD5
09	DES	56 bits	SHA-1
0A	Triple DES	168 bits	SHA-1
2F	AES	128 bits	SHA-1
35	AES	256 bits	SHA-1

The terms used in this table are:
MD5 - Message Digest algorithm
SHA-1 - Secure Hash algorithm
RC2, **RC4** - Rivest encryption
DES - Data Encryption Standard
Triple DES - DES applied three times
AES - Advanced Encryption Standard

The CICS ENCRYPTION system initialization table parameter specifies the level of encryption that CICS should use. Beginning with CICS TS V3.1, the default for the ENCRYPTION parameter is STRONG. This means that CICS can use all of the cipher suites shown in the table (and additional ones if you have a higher release of z/OS) to negotiate with clients.

Using CIPHERS to set a minimum and maximum level

You can set a minimum as well as a maximum encryption level by editing the list of cipher suites in the CIPHERS attribute on the appropriate CICS resource definition:

- URIMAP - for outbound Web service requests
- TCPIPSERVICE - for inbound HTTP

Note: You can also specify CIPHERS on the EXEC WEB OPEN command for outbound HTTP connections.

Setting a minimum encryption level provides you with more control over the security of the SSL connections that you will allow. The list of cipher suites for

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000250

each level of the ENCRYPTION parameter follows. This example shows the cipher suites available with z/OS 1.4 and CICS TS V3.1:

- WEAK - 03060102
- MEDIUM - 0903060102
- STRONG - 0504352F0A0903060102

If you do not want SSL to allow the weaker cipher suites during the SSL negotiation with the partner, then you would set a minimum level by editing the CIPHERS attribute to remove the cipher suites that you do not allow. For example, if you wanted to enforce STRONG and not allow the negotiation to drop to WEAK, you would remove the MEDIUM and WEAK cipher suites and keep the unique STRONG cipher suites. In this case your CIPHERS attribute would have 0504352F0A. If the partner does not support the stronger cipher suites, the SSL handshake will fail and the connection will not be established with the partner.

Important: If the SSL handshake negotiates down to using cipher suite 01 or 02, there is no encryption and data will be transmitted in the clear. It is recommended that you remove 01 and 02 from the list of cipher suites if even a minimum of encryption is required.

Optimizing SSL

Implementing SSL will cause an increase in CPU usage. You should only use SSL for applications that really need encrypted data flows. If you decide that you do need encryption, you can optimize the performance of SSL in your environment by:

- Utilizing the zSeries cryptographic hardware. z/OS 1.6 System SSL uses the CP Assist for Cryptographic Function (CPACF) directly. You can also customize the encryption settings to use only the cipher suites that use ICSF.
- Increasing the value of the CICS system initialization table parameter SSLDELAY so the session IDs remain in the SSLCACHE longer, which will result in only partial SSL handshakes.

  Note: The CICS system initialization table parameter SSLDELAY does not apply to caching across a sysplex.

- Increasing the value of the CICS system initialization table parameter MAXSSLTCBS so there are more S8 TCBs in the SSL pool for the SSL handshake negotiation.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 251/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000251

- Using the CICS SSLCACHE system initialization table parameter to implement SSL caching across a sysplex. If the cache is shared between a number of CICS regions, the throughput of SSL connections will improve.
- Keeping the socket open by coding SOCKETCLOSE NO on the TCPIPSERVICE definition for the PIPELINE. This is the default for HTTP 1.1 persistent sessions and removes the need to perform a full SSL handshake on the second or subsequent HTTP request.
- Only using client authentication by specifying SSL(CLIENTAUTH) on the TCPIPSERVICE definition when you really need your clients to identify themselves with a client certificate. Client authentication requires more network transmissions during the SSL handshake, and more processing by CICS to handle the received certificate.

Setting the user ID on the URIMAP

You can specify a user ID on the URIMAP to be used on behalf of a Web service client. You do this by setting the USERID attribute of the URIMAP definition. It specifies the 1 to 8 character user ID under which the Web services pipeline alias transaction is attached.

Note: If you define and install URIMAP resource definitions explicitly using CEDA, and you are using the CICS Web services assistant to create the Web services, you cannot take advantage of the dynamic installation of URIMAP resources when the PIPELINE resource is installed. This is because the user ID can also be specified on one of the parameters input to the CICS Web services assistant, in which case it would be used to define the USERID attribute of the URIMAP resource when it is created automatically during the PIPELINE scan. If you explicitly code USERID for the URIMAP, and install the URIMAP, it will override the dynamic installation of the URIMAP resource when the PIPELINE resource is installed.

A user ID that you specify in the URIMAP definition is overridden by any user ID that is obtained directly from the client.

Important: If you use a URIMAP definition to set a user ID, there is no authentication of the client's identity. You should only do this when communicating with your own client system, which has already authenticated its users and communicates with the server in a secure environment.

Order of precedence for determining the user ID when using HTTP

It is possible that for a single Web service request transported by HTTP, multiple methods for setting the user ID will be used at the same time. In this event, CICS

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000252

uses the following order of precedence for determining the user ID under which the target business logic program runs:

1. A user ID specified by a message handler, or a SOAP header processing program, that is included in the pipeline that processes the SOAP message. For example, a SOAP header processing program could extract a username from the SOAP message and specify that the CICS task should run with this user ID. See "SOAP message security" on page 226 for more information on how SOAP message security can be used with CICS and HTTP.
2. A user ID obtained from the Web client using basic authentication, or a user ID associated with a client certificate.
3. A user ID specified in the URIMAP definition for the request.
4. The CICS default user ID, if no other can be determined.

7.3.2 WebSphere MQ transport

To control security checking performed by WebSphere MQ (WMQ), you must define *switch profiles*. When a queue manager is started (or when the WMQ REFRESH SECURITY command is issued), WMQ first checks the status of RACF and the MQADMIN class. It sets the subsystem security switch off if it discovers one of these conditions:

- RACF is inactive or not installed.
- The MQADMIN class is not defined to RACF.
- The MQADMIN class has not been activated.

If both RACF and the MQADMIN class are active, WMQ checks the MQADMIN class to see whether any of the switch profiles have been defined. If subsystem security is not required, WMQ sets the internal subsystem security switch off and performs no further checks. The sequence of subsystem security checks is shown in Figure 7-2.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000253

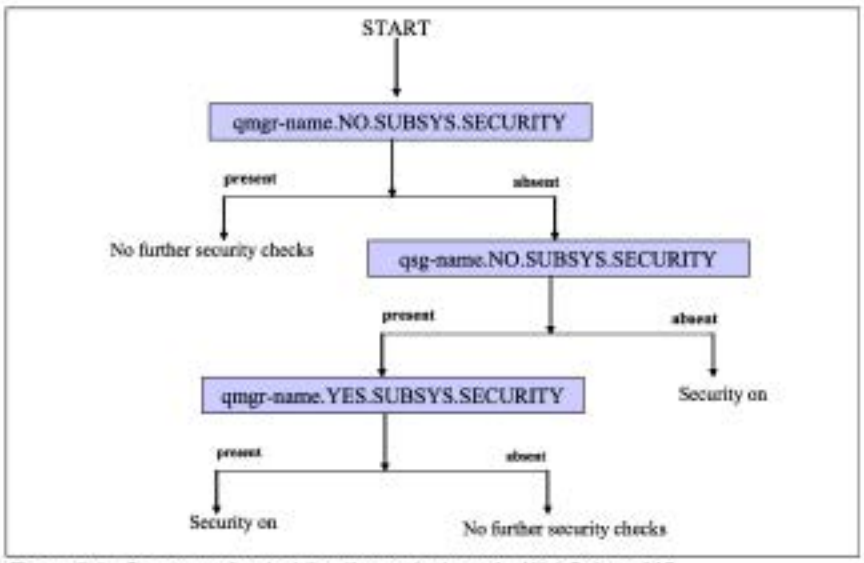


Figure 7-2 Sequence for deciding if security is on for WebSphere MQ

Switch profiles can be set at the queue manager level and at the queue-sharing group level, but the queue manager level is always checked first. If your queue manager is not a member of a queue-sharing group, then no queue-sharing group checks are made. Switch profiles are not subject to any access list checks and are merely used to indicate to WebSphere MQ whether a particular security switch is on or off. A number of switch profiles exist that can be used to control the security checking for your WebSphere MQ environment.

When using WebSphere MQ as the transport mechanism for accessing Web services in CICS, you need to consider the following points:

- The SOAP MQ inbound listener transaction (CPIL) is started by the trigger monitor using the same user ID as the trigger monitor transaction. This user ID must have UPDATE authority to the request queue and the backout queue (if this is specified).
- If AUTH=IDENTIFY is specified in the USERDATA parameter of the WMQ PROCESS definition for CPIL, then the user ID under which CPIL runs must have surrogate authority to allow it to start transactions on behalf of the user IDs in the MQ message descriptors (MQMDs) of the messages.

More information about security for WMQ can be found in *WebSphere MQ Security*, SC34-6588 and in the redbook *WebSphere MQ Security in an Enterprise Environment*, SG24-6814.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 254/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000254

SSL/TLS with WebSphere MQ

SSL/TLS can be used to secure SOAP messages that are transported using WMQ. WMQ supports Version 3.0 of the SSL protocol. You specify the cryptographic algorithms that are used by the SSL protocol by supplying a CipherSpec as part of the channel definition. WMQ also supports Version 1.0 of the Transport Layer Security (TLS) protocol.

See *WebSphere MQ Security*, SC34-6588 for more information on using SSL/TLS with WMQ.

Order of precedence for determining user ID when using WMQ

It is possible that for a single Web service request transported by WMQ, multiple methods for setting the user ID will be used at the same time. In this event, CICS uses the following order of precedence to determine the user ID under which the target business logic program runs:

1. A user ID specified by a message handler, or a SOAP header processing program, that is included in the pipeline that processes the SOAP message

   For example, a SOAP header processing program could extract a username from the SOAP message and specify that the CICS task should run with this user ID. See "Enabling SOAP message security with WMQ" on page 284 for more information on how SOAP message security can be used with CICS and WMQ.

2. A user ID obtained from the MQ message descriptor

   A message can contain message context information, such as a user ID. This information is held in the message descriptor and can be generated by the queue manager when a message is put on a queue by an application, or by the application itself. This allows the receiving application to run with the same identity as the application that put the message on the queue.

3. The CICS default user ID, if no other can be determined

Note: We found that although URIMAP resources are required for service providers using WMQ, the USERID and TRANSACTION attributes were ignored.

7.4 SOAP message security

The first version of the WS-Security specification was proposed by IBM, Microsoft, and VeriSign in April 2002. After the formalization of the April 2002 specification, the specification was transferred to the OASIS consortium:

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 255/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000255

http://www.oasis-open.org

The latest core specification, Web Services Security: SOAP Message Security 1.0 (WS-Security 2004) was standardized in March 2004.

WS-Security provides a foundational set of SOAP message extensions for building secure Web services by defining new elements to be used in the SOAP header for message-level security. It specifies the use of *security tokens, digital signatures*, and *XML encryption* to protect and authenticate SOAP messages. It specifies the use of digital signatures to provide integrity for XML elements in a SOAP message, and it specifies the use of encryption to provide confidentiality for XML elements in a SOAP message. The specification allows you to protect the body of the message or any XML elements within the body or the header. You can give different levels of protection to different elements within the SOAP message.

The advantage of using WS-Security over SSL is that it can provide *end-to-end* message-level security. This means that the message security can be protected even if the message goes through multiple services, called intermediaries. SSL security is considered to be *point-to-point*, and the data may be decrypted prior to reaching the intended recipient.

As illustrated in Figure 7-3, if the service requester identifies itself to the intermediate gateway, and the intermediate gateway identifies itself to the service provider, the target service will normally run with the identity of the intermediate gateway rather than the service requester.

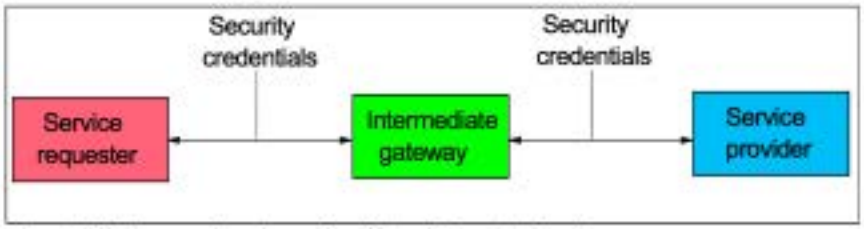


Figure 7-3 Transport-level security with an intermediate gateway

WS-Security addresses this problem by allowing security credentials to be passed within the SOAP message, so that the credentials of the service requester can be passed via an intermediate gateway, and can still be used to identify the requester to the service provider. See Figure 7-4.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 256/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000256

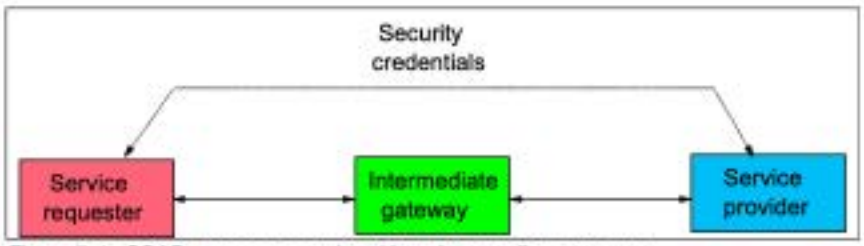


Figure 7-4 SOAP message security with an intermediate gateway

Figure 7-5 shows how a SOAP message can be extended with security data that is used to authenticate the service requester and to protect the message as it passes between the requester and the service provider. The network portion of the diagram could contain any number of intermediate nodes, some of which may not be trusted.

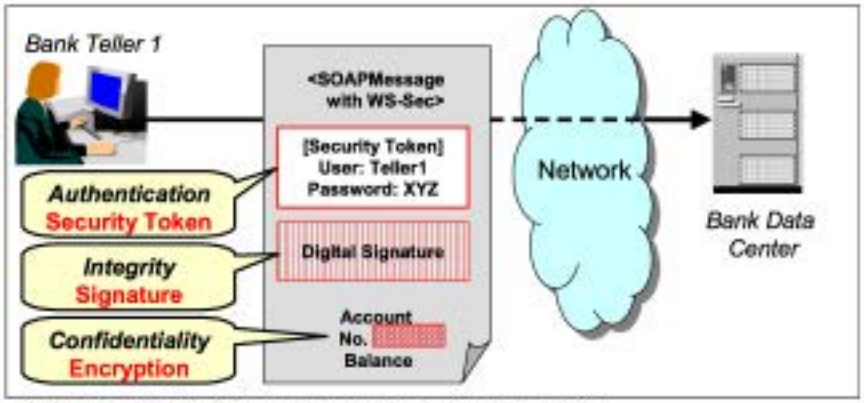


Figure 7-5 An example of a typical scenario with WS-Security

The SOAP message shown in Figure 7-5 contains three pieces of security data:

- A security token used to authenticate and identify user Teller1
- A digital signature to ensure that no one modifies the message while it is in transit without the modification being detected
- An account balance XML element that is encrypted to ensure confidentiality

To read more about the Web services security specifications, refer to:

- Specification: Web Services Security (WS-Security) Version 1.0 (April 2002):
  `http://www.ibm.com/developerworks/webservices/library/ws-secure/`

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 257/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000257

- Web Services Security Addendum (August 2002):

  http://www.ibm.com/developerworks/webservices/library/ws-secureadd.html

- Web Services Security: SOAP Message Security V1.0 (March 2004):

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

Important: When we wrote the first edition of this book, CICS did not provide support for WS-Security. However, it was still possible for a user-written SOAP header processing program to process WS-Security security tokens in CICS. We show how to do this in Chapter 8, "Security scenarios" on page 247.

7.4.1 CICS and SOAP message security

Using WS-Security, you can apply authentication, integrity, and confidentiality at the *message* level. CICS TS V3.1 provides support for WSS: SOAP Message Security through the use of a CICS-supplied message handler, DFHWSSE1, which was shipped by APAR PK22736.

The WS-Security implementation in CICS TS V3.1 contains code derived from the Apache XML Security project. The licensing terms associated with that code mean that the WS-Security implementation in CICS TS V3.1 is not licensed on the same basis as the rest of the CICS TS V3.1 product.

Except for the WS-Security implementation, CICS TS V3.1 is an "ICA Program," licensed under the relevant terms and conditions of the IBM Customer Agreement (ICA) or IBM International Customer Agreement (IICA). The WS-Security implementation is licensed under the terms and conditions of the IBM International Program License Agreement (IPLA). IPLA licensing is widely used in IBM, especially for products on distributed platforms and for one-time charge products on z/OS.

The WS-Security implementation, known as the CICS WS-Security Component, is packaged as a unique FMID with the identifier JCI640W. FMID JCI640W is licensed under the IPLA.

For this reason, the Licensed Program Specification (LPS) for CICS TS V3.1 includes a section explaining that the CICS WS-Security Component has a different licensing basis from the rest of CICS TS V3.1. Specifically, the licensing of the CICS WS-Security Component is addressed by the following three additional paper items:

- The IPLA (multi-language booklet)
- License Information (LI) document for CICS WS-Security Component (multi-language booklet)

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000258

- Proof of Entitlement for CICS WS-Security Component (multi-language sheet of paper)

Deliveries of CICS TS V3.1 after June 2, 2006 contain the additional material. Customers who ordered and received CICS TS V3.1 before that date were sent the material separately, together with the updated LPS.

Important: The WS-Security implementation in CICS Transaction Server for z/OS Version 3.1 is not licensed on the same basis as is used for the rest of the CICS TS V3.1 product.

There are several options available with the CICS WS-Security support, and which ones you choose will depend on the level of security required for the data and the transmission path of the data. The options that you can choose from are:

- Basic authentication

  In service provider mode, CICS can accept a UsernameToken in the SOAP message header for authentication on inbound SOAP messages. The UsernameToken contains a Username element and a Password element. CICS verifies the Username and Password using an external security manager such as RACF. If this is successful, CICS places the Username in container DFHWS-USERID and processes the SOAP message in the pipeline. If CICS is unable to verify the UsernameToken, it returns a SOAP fault message to the service requester.

  Username tokens are not supported on outbound SOAP messages or in service requester mode.

- Signing with X.509 certificates

  In service provider and service requester modes, you can provide an X.509 certificate in the SOAP message header to sign the body of the SOAP message for authentication. An X.509 certificate is an example of a binary security token.

  - Inbound SOAP messages

    To accept binary security tokens from inbound SOAP messages you must import the public key associated with the certificate into RACF and associate it with the keyring that is specified in the KEYRING system initialization parameter for the CICS region.

  - Outbound SOAP messages

    For outbound SOAP messages you need to generate and publish the public key to the intended recipients. The Integrated Cryptographic Service Facility (ICSF) is used to generate private keys.

- Encrypting

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 259/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000259

In service provider and service requestor modes, you can encrypt the SOAP message body using either the Triple DES algorithm or the AES algorithm. It is then included in the message and encrypted using the intended recipient's public key with the asymmetric key encryption algorithm RSA 1.5.

CICS does not support inbound SOAP messages that only have an encrypted element in the message header and no encrypted elements in the SOAP body.

- Signing and encrypting

  In service provider and service requester modes, you can choose to both sign and encrypt a SOAP message. CICS always signs the SOAP message body first and then encrypts it. This provides both message integrity and confidentiality.

Authentication

In this section, we provide an example of a SOAP message with WS-Security used for authentication.

WS-Security provides a general purpose mechanism to associate security tokens with messages for single message authentication. It does not require you to use a specific type of security token. Instead it is designed to be extensible and support multiple security token formats to accommodate a variety of authentication mechanisms. For example, a client might provide proof of identity and proof of a particular business certification.

Example 7-1 shows a sample SOAP message without applying WS-Security. The SOAP message is an Order request for our sample catalog application.

Example 7-1 SOAP message without WS-Security

```
<soapenv:Envelope
     xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
     xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header/>
  <soapenv:Body>
    <p635:DFHOXCHN xmlns:p635="http://www.DFHOXCHN.DFHOXCPS.Request.com">
      <p635:ca_request_id>01ORDR</p635:ca_request_id>
      <p635:ca_return_code>0</p635:ca_return_code>
      <p635:ca_response_message></p635:ca_response_message>
      <p635:ca_order_request>
        <p635:ca_userid>srthstrh</p635:ca_userid>
        <p635:ca_charge_dept>hahhhh</p635:ca_charge_dept>
        <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000260

```
        <p635:ca_quantity_req>1</p635:ca_quantity_req>
        <p635:filler1 xsi:nil="true" />
      </p635:ca_order_request>
    </p635:DFHOXCMN>
  </soapenv:Body>
</soapenv:Envelope>
```

As you can see in Example 7-1, the SOAP message does not have any SOAP headers. We will apply WS-Security by inserting a SOAP security header.

WS-Security defines a vocabulary that can be used inside the SOAP envelope. The XML element <wsse:Security> is the *container* for security-related information. (Note that wsse stands for *Web services security extension*.)

When you use WS-Security for authentication, a security token is embedded in the SOAP header and is propagated from the message sender to the intended message receiver. On the receiving side, it is the responsibility of the server security handler to authenticate the security token and to set up the caller identity for the request.

In Example 7-2 we show the same SOAP message, but this time with authentication. As you can see, we have user name and password information contained in the <UsernameToken> element.

Example 7-2 SOAP message with WS-Security

```
<soapenv:Envelope
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.
     xsd">
      <wsse:UsernameToken>
        <wsse:Username>WEBUSER</wsse:Username>
        <wsse:Password
         Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1
         .0#PasswordText">
           REDBOOKS
        </wsse:Password>
      </wsse:UsernameToken>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <p635:DFHOXCMN xmlns:p635="http://www.ibm.com/DFHOXCMN.JFHOXCPB.Request.com">
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000261

```
      <p635:ca_request_id>01ORDR</p635:ca_request_id>
      <p635:ca_return_code>0</p635:ca_return_code>
      <p635:ca_response_message></p635:ca_response_message>
      <p635:ca_order_request>
        <p635:ca_userid>srthstrh</p635:ca_userid>
        <p635:ca_charge_dept>hbhhhh</p635:ca_charge_dept>
        <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
        <p635:ca_quantity_req>1</p635:ca_quantity_req>
        <p635:filler1 xsi:nil="true" />
      </p635:ca_order_request>
    </p635:DFHOXCMN>
  </soapenv:Body>
</soapenv:Envelope>
```

The <UsernameToken> element of the SOAP message in Example 7-2 contains credentials that can be used to authenticate the user WEBUSER.

The simplest form of security token is the UsernameToken, which is used to provide a user name and password for basic authentication. In "The WS-Security header processing program" on page 267, we show an example of how a header processing program can extract a UsernameToken from a SOAP header, validate the username and password, and set the user ID of the CICS task to the username passed in the header.

A signed security token is one that is cryptographically signed by a specific authority. For example, an X.509 certificate is a signed security token.

Security token usage for WS-Security is defined in separate profiles such as the Username token profile and the X.509 token profile.

To read more about these security token standards, refer to:

- Web Services Security: UsernameToken Profile V1.0 (March 2004):

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

- Web Services Security: X.509 Token Profile V1.0 (March 2004):

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

Implementing SOAP message security in CICS

To implement WS-Security in CICS TS for either a service provider or a service requester, you must include a <wsse_handler> element in the configuration file for the appropriate pipeline. Use the following sub-elements of <wsse_handler> to provide configuration information to DFHWSSE1:

- <dfhwsse_configuration>

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000262

This element can be used in either a service provider or a service requester pipeline. It may contain the following optional elements: each of these is discussed in more detail below:

- `<authentication>`
- `<expect_signed_body/>`
- `<expect_encrypted_body/>`
- `<sign_body>`
- `<encrypt_body>`

Detailed discussion of these optional elements follows.

- `<authentication>`

  Specifies the use of security tokens in the headers of inbound and outbound SOAP messages. It can be used in either a service provider or a service requester pipeline. In a service provider pipeline, the element specifies whether CICS should use the security tokens in an inbound SOAP message to determine the user ID under which work will be processed. In a service requester pipeline, it specifies that CICS should add an X.509 certificate to the security header for outbound SOAP messages.

  The `<authentication>` element has two attributes: *trust* and *mode*. These attributes determine whether asserted identity is used and the combination of security tokens used in a SOAP message. The *trust* attribute can be set to either none, basic or signature. The *mode* attribute can also be set to either none, basic, or signature. For more information on the meaning and valid combinations of these attributes, refer to the *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SG34-6458.

  Asserted identity allows a trusted user to assert, or declare, that work should run under a different identity (the *asserted identity*), without the trusted user having the credentials associated with that identity. Messages contain a *trust token* and an *identity token*. The adjectives *trust* and *identity* indicate how the token is used rather than the kind of token. A trust token, for example, could be a UsernameToken or an X.509 token. The trust token is used to check that the sender has the correct permissions to assert identities, and the identity token holds the asserted identity (user ID) under which the request is to run.

  Note: If you use asserted identity, it requires that the service provider trusts the requester to make this assertion. In CICS, the trust relationship is established with security manager surrogate definitions: the requesting identity must have the correct authority to start work on behalf of the asserted identity.

  The `<authenticate>` element can contain the following elements:

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 263/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000263

- <certificate_label>

  Optional. Specifies the label associated with an X.509 digital certificate ignored in a service provider pipeline

- <suppress/>

  Optional. For a service provider, the handler will not use any security tokens in the message to determine under which user ID to run. For a service requester, the handler will not add to the SOAP message any of the security tokens required for authentication

- <algorithm>

  Specifies the URI of the signature algorithm. CICS currently supports the signature algorithms for inbound SOAP messages shown in Table 7-2.

Table 7-2 Signature algorithms for inbound SOAP messages

Algorithm	URI
Digital Signature Algorithm with Secure Hash Algorithm 1 (DSA with SHA1)	http://www.w3.org/2000/09/xmldsig#dsa-sha1
Rivest-Shamir-Adleman algorithm with Secure Hash Algorithm 1 (RSA with SHA1)	http://www.w3.org/2000/09/xmldsig#rsa-sha1

- <expect_signed_body/>

  Indicates that the <body> of the inbound message must be properly signed. If it is not, CICS rejects the message with a security fault.

- <expect_encrypted_body/>

  Indicates that the <body> of the inbound message must be properly encrypted. If it is not, CICS rejects the message with a security fault.

- <sign_body>

  Directs DFHWSSE1 to sign the body of outbound SOAP messages, and provides information regarding how the messages are to be signed. It can be used in either a service provider or a service requester pipeline. It contains the following elements:

  - <algorithm>

    Specifies the URI of the algorithm used to sign the body of the SOAP message.

    CICS currently supports the following signature algorithm for outbound SOAP messages:

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000264

- Rivest-Shamir-Adleman algorithm with Secure Hash Algorithm 1 (RSA with SHA1), which is specified using the URI http://www.w3.org/2000/09/xmldsig#rsa-sha1

- <certificate_label>

  Specifies the label associated with an X.509 digital certificate. The digital certificate should contain the private key since this was used to sign the message. The public key associated with the private key is then sent in the SOAP message, which allows the signature to be validated.

- <encrypt_body>

  Directs DFHWSSE1 to encrypt the body of outbound SOAP messages, and provides information regarding how the messages are to be encrypted. It can be used in both a service provider and service requester pipeline. It contains the following elements:

  - <algorithm>

    Specifies the URI identifying the algorithm used to encrypt the body of the SOAP message. CICS currently supports the encryption algorithms shown in Table 7-3.

Table 7-3 Encryption algorithms

Algorithm	URI
Triple DES in cipher block chaining mode	http://www.w3.org/2001/04/xmlenc#tripledes-cbc
AES with a key length of 128 bits in cipher block chaining mode	http://www.w3.org/2001/04/xmlenc#aes128-cbc
AES with a key length of 192 bits in cipher block chaining mode	http://www.w3.org/2001/04/xmlenc#aes192-cbc
AES with a key length of 256 bits in cipher block chaining mode	http://www.w3.org/2001/04/xmlenc#aes256-cbc

    <certificate_label>

    Specifies the label associated with an X.509 digital certificate. The digital certificate should contain the public key of the intended recipient of the SOAP message so that it can be decrypted with the private key when the message is received.

Example 7-3 shows a <wsse_handler> element with all of the optional elements present. You would add this to your configuration file for the pipeline.

Example 7-3 <wsse_handler>

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000265

```
<wsse_handler>
 <dfhwsse configuration version="1">
   <authentication trust="signature" mode="basic">
     <certificate_label>AUTHCERT03</certificate_label>
     <suppress/>
   </authentication>
   <expect_signed_body/>
   <expect_encrypted_body/>
   <sign_body>
     <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
     <certificate_label>SIGCERT01</certificate_label>
   </sign_body>
   <encrypt_body>
     <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
     <certificate_label>ENCCERT02</certificate_label>
   </encrypt_body>
 </dfhwsse configuration>
</wsse_handler>
```

Example 7-4 shows the pipeline configuration file basicsoap11provider.xml for the EXPIPE01 service provider pipeline associated with the sample catalog application.

Example 7-4 CICS-supplied sample pipeline configuration file basicsoap11provider.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
   xmlns="http://www.ibm.com/software/htp/cics/pipeline"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Example 7-5 shows how you would modify the pipeline configuration file to add the <service_handler_list> and <wsse_handler> elements to implement WS-Security. CICS will read the pipeline configuration file and when it finds the <wsse_handler> element it will load program DFHWSSE1 from library SDFHWSLD in your DFHRPL concatenation to process the security information. For more information on the elements for the pipeline configuration file, and

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000266

which ones are contained by other elements (high-level structure diagrams), refer to the most current version of *CICS Transaction Server for z/OS V3.1 Web Services Guide*, SG24-6458.

Example 7-5 <wsse_handler> element added to configuration file basicsoap11provider.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<provider_pipeline
    xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipelineprovider.xsd">
  <service>
    <service_handler_list>
      <wsse_handler>
        <dfhwsse configuration version="1">
          <authentication trust="signature" mode="basic">
            <certificate_label>SUICERT03</certificate_label>
            <suppress/>
          </authentication>
          <expect_signed_body/>
          <expect_encrypted_body/>
          <sign_body>
            <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
            <certificate_label>SIGCERT01</certificate_label>
          </sign_body>
          <encrypt_body>
            <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm>
            <certificate_label>ENCCERT02</certificate_label>
          </encrypt_body>
        </dfhwsse configuration>
      </wsse_handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal handler>
  </service>
  <appHandler>DFHPITP</appHandler>
</provider_pipeline>
```

If CICS is the service provider, CICS will decrypt any inbound encrypted SOAP message automatically when it processes the message, provided you have the <wsse_handler> element in the pipeline configuration file. The security header in the received message provides all of the information needed for CICS to decrypt it. In other words, the <encrypt_body> and <sign_body> elements do not need to be specified in the provider pipeline configuration file in order to decrypt the

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000267

inbound SOAP message. But you can (and probably will want to) include the <encrypt> <body> or <sign> <body>, or both, in the provider pipeline configuration file if you want to encrypt or sign the reply body sent back to the requester. This is what we have shown in Example 7-5.

7.4.2 WebSphere and SOAP message security

WebSphere Application Server V6.1 is based on the implementation of WS-Security in the following OASIS specification and profiles:

- WS-I Basic Security Profile

  http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html

- Web Services Security: SOAP Message Security 1.0 (March 2004):

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf

- Web Services Security: UsernameToken Profile 1.0 (March 2004).

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0.pdf

- Web Services Security: X.509 Certificate Token Profile V1.0 (March 2004)

  http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-1.0.pdf

WebSphere Application Server uses the J2EE 1.4 Web services deployment model to implement WS-Security. The Web services security constraints are specified in the IBM extension of the Web services deployment descriptors and bindings. The Web services security runtime enforces the security constraints specified in the deployment descriptors. One of the advantages of this deployment model is that you can define the Web services security requirements outside of the application business logic. With the separation of roles, the application developer can focus on the business logic, and the security expert can specify the security requirement.

Figure 7-6 shows the high-level architecture model that is used to secure Web services in WebSphere Application Server.

Document Id: [illegible] Page 268(5)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000268

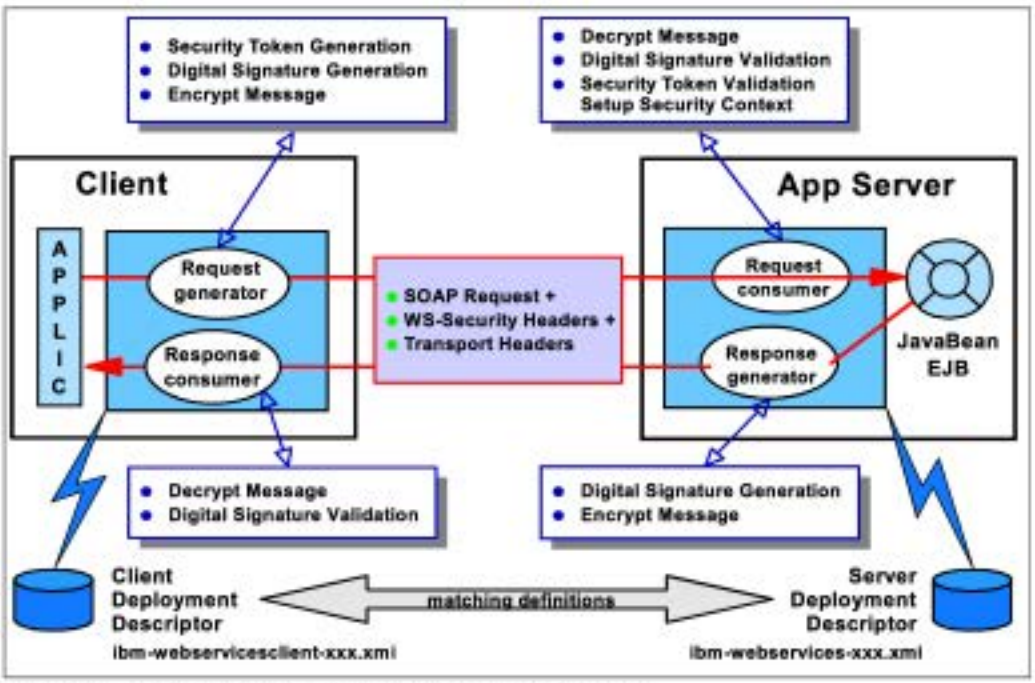


Figure 7-6 WebSphere Application Server support for WS-Security

As shown in the figure, there are two sets of configurations on both the client side and the server side.

- **Request generator**

  This client-side configuration defines the Web services security requirements for the outgoing SOAP message request. These requirements might involve generating a SOAP message request that uses a digital signature, incorporates encryption, and attaches a security token.

- **Request consumer**

  This server-side configuration defines the Web services security requirements for the incoming SOAP message request. These requirements might involve verifying that the required integrity parts are digitally signed, verifying the digital signature, verifying that the required confidential parts were encrypted by the request generator, decrypting the required confidential parts, validating the security token, and verifying that the security context is set up with the appropriate identity.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 269/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000269

- **Response generator**

  This server-side configuration defines the Web services security requirements for the outgoing SOAP message response. Those requirements might involve generating the SOAP message response with Web services security, including digital signature, and encrypting and attaching a security token, if necessary.

- **Response consumer**

  This client-side configuration defines the Web services security requirements for the incoming SOAP response. The requirements might involve verifying that the integrity parts are signed and the signature is verified; verifying that the required confidential parts are encrypted and that the parts are decrypted; and validating the security token, if necessary.

The Web services security requirements that are defined in the request generator must match the request consumer. The requirements that are defined in the response generator must match the response consumer. Otherwise, the request or response is rejected because the Web services security constraints cannot be met by the request consumer and response consumer.

The format of the Web services security deployment descriptors and bindings are IBM proprietary. However, the following tools are available to edit the deployment descriptors and bindings.

- Rational Application Developer V6.0 (RAD)

  In Chapter 8, "Security scenarios" on page 247 we show how we configured WS-Security for our sample catalog manager application using RAD.

- Application Server Toolkit V6.1 (AST)

  In Chapter 9, "Security scenarios using CICS WS-Security support" on page 295 we show how we configured WS-Security using the AST.

7.5 Comparison of transport versus SOAP message security

We have shown in this chapter that it is possible to implement Web services security at two levels: the transport level and the SOAP message level. If your Web services environment is simple (for example, it does not span multiple nodes), a security solution based on transport-level security alone may be all that you need. For more complex scenarios, however, it may not be enough on its own.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000270

In this section we provide general guidelines to help you decide what type of security solution to implement.

- You might chose to use only transport-level security to secure your CICS Web services environment when:
  - No intermediaries are used in the Web service environment or, if there are intermediaries, then you can guarantee that once the data is decrypted, it cannot be accessed by an untrusted node or process

    The transport is only based on HTTP.
  - The Web services client is a stand-alone Java program

    WS-Security can only be applied to clients that run in a Web services environment that supports the WS-Security specification (for example, WebSphere Application Server).
- You might chose to use WS-Security (possibly in addition to transport-level security) when:
  - Intermediaries are used, some of which may be untrusted.

    Security credentials that flow in the SOAP message can pass through any number of intermediaries. Protecting confidential information in the actual SOAP message may avoid the overhead of encrypting and decrypting via SSL at every intermediary node.

    Furthermore, an intermediary may be able to provide an authentication service to CICS, such that the intermediary server authenticates the Web service client and then flows an *asserted* identity to CICS.
  - Multiple transport protocols are used.

    WS-Security works across multiple transports and is independent of the underlying transport protocol.
  - The Web service partners support WS-Security and a general decision has been taken to flow security tokens in accordance with the WS-Security specification.
  - You choose to implement your own security procedures and processing by writing a custom message handler program that can process secure SOAP messages in the pipeline.

7.6 Securing CICS Web services using the service integration bus

WebSphere Application Server provides the ability to use the *service integration bus* as an intermediary between service requesters and service providers.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000271

allowing you to control the flow, routing, and transformation of messages through mediations and JAX-RPC handlers (see Chapter 5, "Connecting CICS to the service integration bus" on page 129).

The bus provides a flexible way to expose and call services located in an intranet from the Internet (and vice versa), while also providing mechanisms for protocol switching and security, including extensive support for the WS-Security specification.

Figure 7-7 illustrates how the bus could be used to enable service requesters to access CICS Web services via a *gateway service*. See "Creating a gateway service on the bus" on page 139 for information on how we enabled the inquireSingle Web service of the CICS sample catalog application to be invoked via a gateway service.

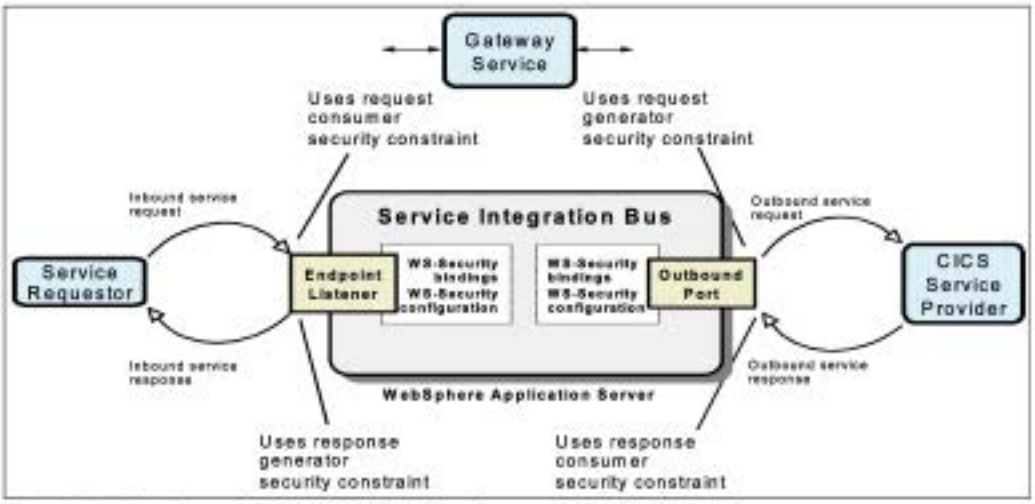


Figure 7-7 Securing a CICS Web service using the service integration bus

One advantage of such a configuration is that you can configure the bus for secure transmission of SOAP messages using tokens, digital signatures, and encryption, in accordance with the WS-Security specification. For each service, you can select the security settings that are applied between the service requester and the gateway service, and between the gateway service and the service provider (in this case, CICS).

WS-Security security constraints can be configured for the following:

- Request consumer - used on inbound requests from a client to a service.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 272/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000272

- Request generator - used when generating outbound requests from a service to a target Web service.
- Response consumer - used on outbound responses from a target Web service to a service.
- Response generator - used when generating inbound responses from a service to a client.

The bus can also invoke Web services that include https:// in their addresses. This provides message integrity and confidentiality while the message is transmitted between the bus and the CICS region.

7.7 Identity assertion

Identity assertion is an authentication mechanism that is applied among three parties: a client, an intermediary server, and a target server.

- A request message is sent to an intermediary server with a client's security token.
- The intermediary server authenticates the client and transfers the client's request message and identity to the target server with the *intermediary's* security token.

Identity assertion is an extended security mechanism supported by WebSphere Application Server Version 6.0 and the service integration bus. There are several options for sending the client's identity with the intermediary's token to the target server.

It is possible to use identity assertion to secure a CICS Web service via the bus. For example, for the Gateway service shown in Figure 7-7, we could:

- Define a security requirement for the request consumer such that the requester must provide a binary security token, such as an X.509 certificate.
- Configure the bus (WebSphere Application Server) to map the certificate to a user identity.
- Define a security requirement for the request generator such that the bus propagates the service requester's identity to CICS in a UsernameToken.

The bus must establish a trust relationship with the CICS region by authenticating itself and then by being recognized as a trusted partner of the CICS region. This can be done using one of two different modes:

Trust token	The bus sends a trust token to CICS.
Presumed trust	Trust is established at the transport level rather than at the SOAP message level.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000273

7.7.1 Trust token model

In this model, SOAP messages that flow between the bus and CICS contain a trust token and an identity token. The trust token is used to check that the sender has the correct permissions to assert identities, and the identity token holds the asserted identity (user ID) under which the request is to run. When using the CICS-provided support for WS-Security, a trust token can be a UsernameToken or an X.509 token.

The advantage of the trust token model is that it is independent of the mechanism used to transport the SOAP messages. The disadvantage, however, is the overhead of validating the trust token for each SOAP request.

7.7.2 Presumed trust model

In this model, SOAP messages that flow between the bus and CICS contain only an identity token. The trust relationship between the bus and CICS must be established using a transport-based mechanism such as SSL client authentication.

The CICS-provided support for WS-Security does not support the presumed trust model. To implement such a model, you need to write a custom handler. The user-written header processing program CIWSSECS described in "Configuring the service provider" on page 288 is an example of such a custom handler.

The advantage of the presumed trust model is that the trust established between the bus and CICS can be persistent (for example, by using SSL persistent connections) and does not need to re-established for each SOAP message. The disadvantage, however, is that it is dependent on the mechanism used to transport the SOAP messages.

Recommendation: The presumed trust model offers significant performance advantages over the trust token model.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000274

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000275

8

Security scenarios

In this chapter we outline several security scenarios that demonstrate how you can secure a CICS Web services environment. We start with an explanation of how we prepared the system and the settings that we used for the basic security configuration of our CICS system. We then provide step-by-step security configuration for a number of scenarios, including:

- Setting the user ID on a URIMAP resource definition
- Enabling SOAP message security with HTTP
- Enabling SSL/TLS
- Enabling SOAP message security with WebSphere MQ

Important: The SOAP message security scenarios documented in this chapter were tested on a version of CICS TS V3.1 which did not have the CICS WS-Security PTFs installed. See Chapter 9, "Security scenarios using CICS WS-Security support" on page 295 for scenarios using the CICS supplied WS-Security support.

We show how we configured both CICS and WebSphere Application Server for these security scenarios.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net

Page 276/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000276

8.1 Preparation

Security techniques for securing Web services were discussed in Chapter 7, "Securing Web services" on page 213. In this chapter, we describe the following security scenarios:

- Setting the user ID on a URIMAP definition

  The user ID is then used for Web service requests that match the URIMAP. No authentication of the client's credentials takes place. See "Setting the user ID on a URIMAP definition" on page 252 for details of this scenario.

- Enabling SOAP message security with HTTP

  In this scenario, we show how a header processing program in a service provider pipeline can extract a WS-Security UsernameToken from a SOAP header, validate the username and password, and set the user ID of the CICS task to the username passed in the header. This scenario is described in "Enabling SOAP message security with HTTP" on page 257.

- Enabling SSL/TLS

  In this scenario, we show how SSL/TLS can be used in combination with SOAP message security to secure a connection between WebSphere Application Server and CICS. See "Enabling SSL/TLS" on page 273.

- Enabling SOAP message security with WebSphere MQ (WMQ)

  In this scenario, we show how a header processing program in a service requester pipeline can attach a WS-Security header to a SOAP message, which is then sent, using WMQ, to a CICS service provider. This scenario is described in "Enabling SOAP message security with WMQ" on page 284.

8.1.1 Software checklist

The software we used is listed in Table 8-1.

Table 8-1 Software used in the security scenarios

Windows	z/OS
Internet Explorer V6.0	z/OS V1.6
Windows 2000 SP4	CICS Transaction Server V3.1*
IBM WebSphere Application Server - ND V6.0.2.0	RACF V1.6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000277

Windows	z/OS
Our J2EE applications ▸ CatalogSec.ear Catalog manager service requester application with no security enabled ▸ CatalogSec_WS-Security.ear Catalog manager service requester application with WS-security enabled ▸ CatalogSec_WS-Security_HTTPS.ear Catalog manager service requester application with WS-security and HTTPS enabled	Our user-written CICS programs ▸ CIWSMSGH (message handler program) ▸ CIWSSECH (header processing program) ▸ CIWSSECR (header processing program) ▸ CIWSSECS (header processing program) ▸ MYPARSER (COBOL XML parsing program)

a. The version of CICS TS V3.1 used in these security scenarios does not include the CICS support for WS-Security which was introduced later in PTFs UK15271 and UK15261.

8.1.2 Definition checklist

The definitions we used are listed in Table 8-2.

Table 8-2 Settings used in the security scenarios

Value	CICS TS	WebSphere Application Server
IP name	mvsg3.mop.ibm.com	cam21-pc3.mop.ibm.com
IP address	9.100.193.167	9.120.198.171
TCPIP port	14321	9080
SSL/TLS port	14323	
Jobname	CIWSS3C1	
APPLID	ASPOS3C1	
TCPIPSERVICE	S3C1	
Provider PIPELINE	PIPE1	
Configuration files	ITSO.7PX6.soapprovider.xml ITSO.7PX6.soaprequester.xml ITSO.7200.soapprovider.disabled.xml	
MQ queue manager	MQS3	

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000278

Value	CICS TS	WebSphere Application Server
URIMAP	SECPORDR SECICATA SECISING	

The user IDs we used in our configuration are listed in Table 8-3.

Table 8-3 User IDs

Value	CICS TS
CICS region user ID	CIWS3D
CICS default user ID	CICSUSER
User IDs to which we wish to permit access	CIWSNW WEBAS3

Tip: For the examples described in this book, we permit access to single user IDs. In a production environment, you will probably create a group of users requiring common access. Once a group is built, you can permit access to the group. This allows users' access to be controlled by the group to which they belong, rather than by individual permissions. This simplifies the security definitions required.

8.2 Basic security configuration

First we discuss our basic security configuration, taking a CICS region with no security and configuring it to enable transaction security (we do not implement other types of CICS security such as resource security and command security). We document the system initialization table parameters necessary to set up basic CICS security and then we test our basic security configuration.

For detailed information on CICS security, see *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454.

Because the INQC and INQS transactions are browse-only transactions, we chose not to secure them. The ORDR transaction updates the database, so we decided to protect it from unauthorized use.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 279/9

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000279

Our starting point is a CICS region with no security configured. This means that all transactions run under the CICS default user ID, as shown in Figure 3-12 on page 93.

8.2.1 Setting up basic security configuration

We configured our CICS region with security prefixing, transaction security, and surrogate user security active using the following SIT parameters:

- SEC=YES
- SECPRFX=YES
- XTRAN=YES
- XUSER=YES

SEC=YES was specified to indicate that we wanted RACF services to control access to CICS resources.

We used security prefixing (SECPRFX=YES) in our CICS region, which prevents our RACF security profiles from affecting other CICS regions. This is useful in a production environment since it means all security profiles are unique to an individual region, however, it can mean more work for the security administrator because more profiles must be defined.

XTRAN=YES was specified so CICS would control who could start transactions and XUSER=YES specifies that CICS is to perform surrogate user checking.

8.2.2 Testing the basic security configuration

After enabling security in our CICS region, we attempted to place an order with the sample catalog application. The Web browser received an error message indicating that the use of some transaction is forbidden (Figure 8-1).

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000280

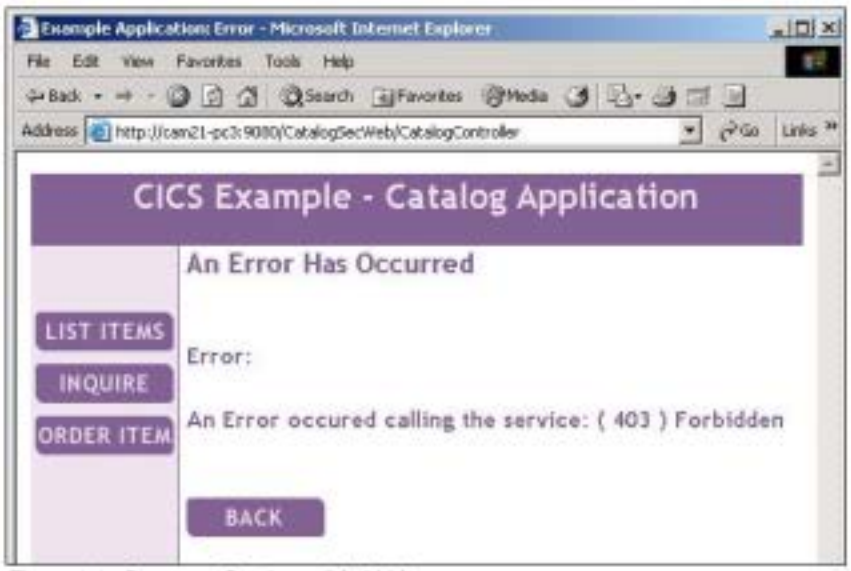


Figure 8-1 Browser - Service call forbidden

Example 8-1 shows the security violation highlighting that the CICS default user ID does not have access to the CPIH transaction.

Example 8-1 CICSUSER security violation messages running CPIH

```
DFHXS1111 11/24/2005 12:33:04 A6P0S3C1 CPIH Security violation by user CICSUSER for resource
          CIWS3D.CPIH in class TCICSTRN. SAF codes are (X'00000008',X'00000000'). ESM codes
          are (X'00000008',X'00000000').
DFHMB0361 11/24/2005 12:33:04 A6P0S3C1 An attempt to attach a CICS Web alias transaction for
          userid CICSUSER has failed because the user is not authorized to execute transaction
          CPIH. Host IP address: 9.100.193.167. Client IP address: 9.100.199.171.
          TCPIPSERVICE: S3C1.
DFHAC2003 11/24/2005 12:33:04 A6P0S3C1 Security violation has been detected term id = ????,
          trans id = CPIH, userid = CICSUSER.
```

8.3 Setting the user ID on a URIMAP definition

With this option the security identity used for the transaction is based on a statically defined user ID. We achieved this preset identification by using a URIMAP resource definition in which we specified a valid user ID (CIWSNW) in the USERID parameter. This is shown in Figure 8-2.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 281/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000281

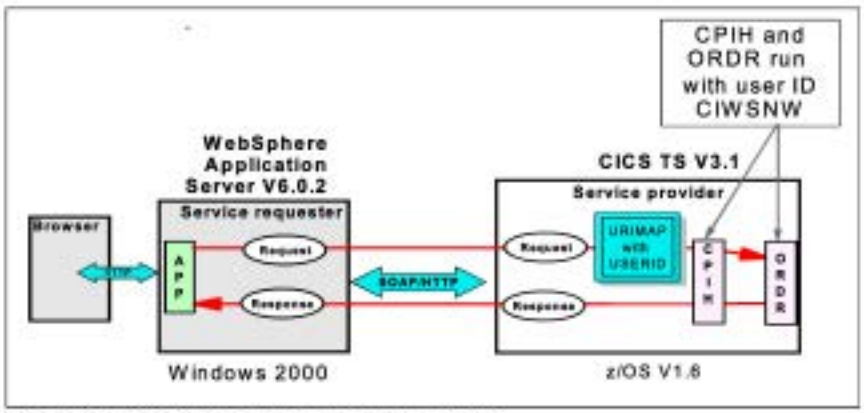


Figure 8-2 Setting user ID on a URIMAP scenario

At this stage, the only security check is to validate whether or not the user ID specified in the URIMAP attribute (CIWSNW) has permission to execute both the CPIH and ORDR transactions.

8.3.1 Defining the URIMAP

We used the following command to define a URIMAP resource:

```
CEDA DEFINE URIMAP(SECPORDR) GROUP(S3C1)
```

We then set the attributes as shown in Figure 8-3 and Figure 8-4.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000282

```
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0640
   CEDA  Define Urimap( SECPORDR )
    Urimap          : SECPORDR
    Group           : SJCI
    Description     ===>
    STatus          ===> Enabled           Enabled | Disabled
    USAge           ===> Pipeline          Server | Client | Pipeline
   UNIVERSAL RESOURCE IDENTIFIER
    SCheme          --> HTTP               HTTP | HTTPS
    HOST            --> *
    (Lower Case)    -->
    PAth            --> /exampleApp/placeOrder
    (Mixed Case)    -->
                    ===>
                    ===>
 ASSOCIATED CICS RESOURCES
    TCpipservice    ===>
 +  ANalyzer        --> No                 No | Yes
                                                      SYSID=SJCI APPLID=A6P0SJCI
```

Figure 8-3 CEDA DEFINE URIMAP (page 1 of 2)

```
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0640
   CEDA  Define Urimap( SECPORDR )
 +  COnverter     ===>
    TRansaction   ===> CPIH
    PRogram       -->
    PIpeline      --> PIPE1
    WebService    --> placeOrder                                  (Mixed Case)
   SECURITY ATTRIBUTES
    USErid        --> CICSUSR
    Ciphers       -->
    CErtificate   ===>                                            (Mixed Case)
   STATIC DOCUMENT PROPERTIES
    MediaType     ===>
    (Lower Case)
    CHaracterset  -->                                             (Mixed Case)
    HOSTCodepage  -->
    TEmplatename  -->
    (Mixed Case)
 +  H-sfile        :
                                                      SYSID=SJCI APPLID=A6P0SJCI
```

Figure 8-4 CEDA DEFINE URIMAP (page 2 of 2)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000283

We used the following values:

- URIMAP name was set to SECPORDR
- USAGE was set to PIPELINE, indicating that it is to be used for incoming Web service requests.
- SCHEME was set to HTTP because no transport security is being used.
- HOST attribute was set to the wildcard (*), meaning that the URIMAP definition matches on any host name
- PATH was set as /exampleApp/placeOrder. Together with the HOST and SCHEME values this means that the URIMAP resource caters to any Web service request originating from http://*/exampleApp/placeOrder.
- TRANSACTION is automatically defaulted to CPIH by CICS when USAGE(PIPELINE) is specified.
- PIPELINE was set to PIPE1, the name of the pipeline for which this URIMAP definition is being made.
- WEBSERVICE can only be specified when USAGE(PIPELINE) is in effect and specifies the name of the Web service to be invoked. We set the value to placeOrder
- USERID was set with the value of CICSNW, the user ID under which we have decided to execute the ORDR transaction.
- All other values were allowed to default

Two other URIMAP resources were defined with similar definitions. First SECPORDR was copied as SECICATA and as SECISING, then these resource definitions were altered as shown in Example 8-2.

Example 8-2 Creating URIMAP definitions for SECICATA and SECISING

```
CEDA COPY URIMAP(SECPORDR) GROUP(SSC1) AS(SECICATA)
CEDA COPY URIMAP(SECPORDR) GROUP(SSC1) AS(SECISING)
CEDA ALTER URIMAP(SECICATA) GROUP(SSC1) PATH(/exampleApp/inquireCatalog)
       WEBSERVICE(inquireCatalog)
CEDA ALTER URIMAP(SECISING) GROUP(SSC1) PATH(/exampleApp/inquireSingle)
       WEBSERVICE(inquireSingle)
```

This ensures that the transactions INQC, INQS, and ORDR all execute with the same user ID CICSNW.

Example 8-3 shows the RACF commands that we used to permit the CICS region user ID (CIWS3D) to start transactions (such as ORDR) with the user ID CIWSNW.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000284

Example 8-3 RACF commands to allow CIWS3D to act as surrogate for CIWSNW

```
RDEFINE SURROGAT CICSUSER.DFHSTART UACC(NONE) OWNER(CICSUSER)
PERMIT CICSUSER.DFHSTART CLASS(SURROGAT) ID(CICS3D) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

For more information regarding surrogate user security, see *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454.

8.3.2 Permitting access to user ID CICSNW

In "Writing the message handler program" on page 80 we explained how the message handler program CIWSMSGH is used to change the default transaction ID (CPIH) to a transaction ID based on the Web service request in the DFHWS-WEBSERVICE container. Example 8-4 shows the RACF commands that we used to limit access to the ORDR transaction to a single user ID (CIWSNW).

Example 8-4 RACF commands to allow access to CPIH and ORDR transactions

```
RDEFINE GCICSTRN CICS3D ADDMEM(CICS3D.CPIH) UACC(NONE) OWNER(IBMUSER)
RALTER  GCICSTRN CICS3D ADDMEM(CICS3D.ORDR)
PERMIT CICS3D CLASS(GCICSTRN) ID(CICSWNW) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

In Example 8-4 we:

- Define the transaction group CIWS3D and add the CPIH and ORDR transactions to the group
- Permit access to the group for user ID CIWSNW
- Refresh the RACF CICS transaction class TCICSTRN

8.3.3 Testing user ID on URIMAP resource definition

With these definitions installed, both the CPIH and ORDR transactions now execute with the user ID as supplied in the URIMAP. This is shown in Figure 8-5.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000285

```
INQUIRE TASK
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tas(0000050) Tra(CEMT) Fac(G312) Sus Ter Pri( 255 )
    Sta(TO) Use(CICSAA  ) Dsn(BD=61FA84DB3006)
 Tas(0000052) Tra(CPTH)           Sus Tas Pri( 001 )
    Sta(L ) Use(CICSHW  ) Lsn(BD=617FF2220DF0) Hty(G2CFAN01)
 Tas(0000053) Tra(ORDR)           Sus Tas Pri( 001 )
    Sta(L ) Use(CICSHW  ) Lsn(BD=617FF553C4774) Hty(BD=     )
 Tas(0000055) Tra(CEUF) Fac(E342) Sus Ter Pri( 001 )
    Sta(SD) Use(CICSAA  ) Dsn(BD-617FF7031E9B0) Hty(ZCIOWA: )
```

Figure 8-5 ORDR executing with preset user ID

8.4 Enabling SOAP message security with HTTP

In Section 7.4 "SOAP message security" on page 226, we explained how WS-Security extends the SOAP specification by defining new elements to be used in the SOAP header for message-level security. We also provided an outline of how WebSphere Application Server supports WS-Security, including support for generating security tokens that are passed in the WS-Security SOAP header. This scenario is shown in Figure 8-6 on page 258.

In this section we show how a simple UsernameToken security token, passed in a Web service call from WebSphere Application Server, can be processed in CICS by a user-written header processing program. The header processing program extracts the UsernameToken from the SOAP header, validates the username and password, and sets the user ID of the CICS task to the username passed in the header.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000286

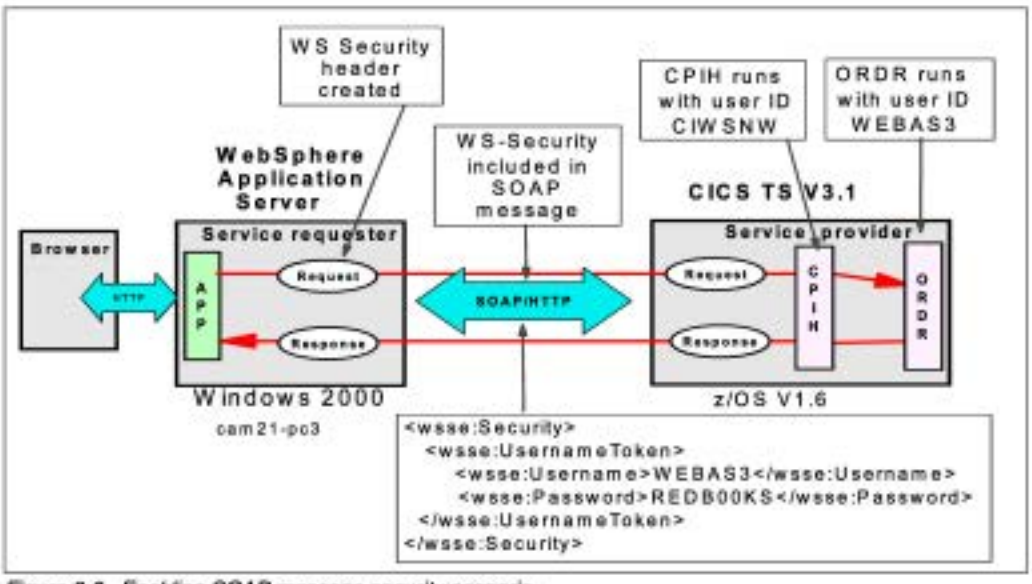


Figure 8-6 Enabling SOAP message security scenario

8.4.1 Configuring the service requester

In "Installing the service requester" on page 87 we showed how we installed our sample Web service client in WebSphere Application Server. The service requester setup tasks that we cover here are as follows:

- Defining the WebSphere WS-Security constraint for the Web service client
- Re-deploying the Web service client application

WebSphere WS-Security configuration files

The WebSphere WS-Security constraints are defined in the IBM extension of the J2EE Web services deployment descriptor. There are four configuration files: application-level deployment descriptor extensions for a client and a server, and binding files for a client and a server (Figure 8-7).

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 287/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000287

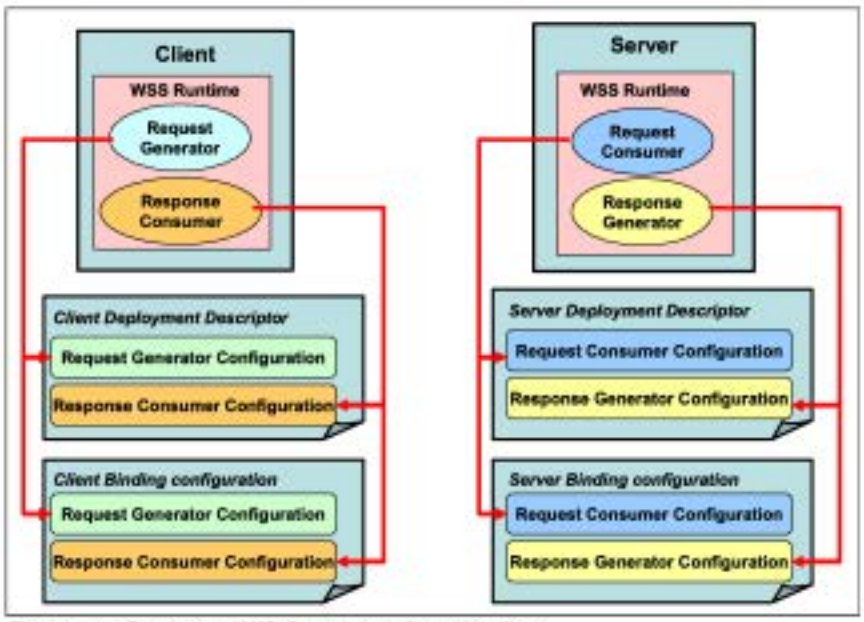


Figure 8-7 Structure of WS-Security configuration files

The configuration files are:

- Client deployment descriptor extension file - includes request generator and response consumer constraints:

  ibm-webservicesclient-ext.xmi

- Client binding configuration file - includes how to apply request generator and response consumer constraints:

  ibm-webservicesclient-bnd.xmi

- Server deployment descriptor extension file - includes request consumer and response generator constraints:

  ibm-webservices-ext.xmi

- Server binding configuration file - includes how to apply request consumer and response generator constraints:

  ibm-webservices-bnd.xmi

Note: The *.xmi* in the file names stands for *XML metadata interchange*.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineVictory.net
Page 288/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000288

The deployment descriptor extension files specify *what* security constraints are required, for example, what type of security token and whether to sign the message. The binding files specify *how* to apply the required security constraints defined in the deployment descriptor extension, for example, which security token is inserted and which keys are used for signing.

These deployment descriptor extension and binding files define the application-level security constraints and they belong to each application.

For our scenario, we needed to configure the client deployment descriptor and the client binding only, since our service provider is running in CICS and not in WebSphere Application Server.

Editing the client configuration

To configure our Web service client to use WS-Security, we used Rational Application Developer V6.0 (RAD).

Note: You can also use the Application Server Toolkit to configure WS-Security for WebSphere applications.

We imported the client application archive CatalogSec.ear into RAD. We then expanded the Dynamic Web Project (CatalogSecWeb) in the Project Explorer and opened (double-clicked) the deployment descriptor.

After the Deployment Descriptor Editor opened, we selected the **WS Extension** page. The WS Extension page is used for editing the client's deployment descriptor extension file, so you can specify what security is required. Figure 8-8 shows the WS Extension page in the Deployment Descriptor Editor.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net

Page 289/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000289

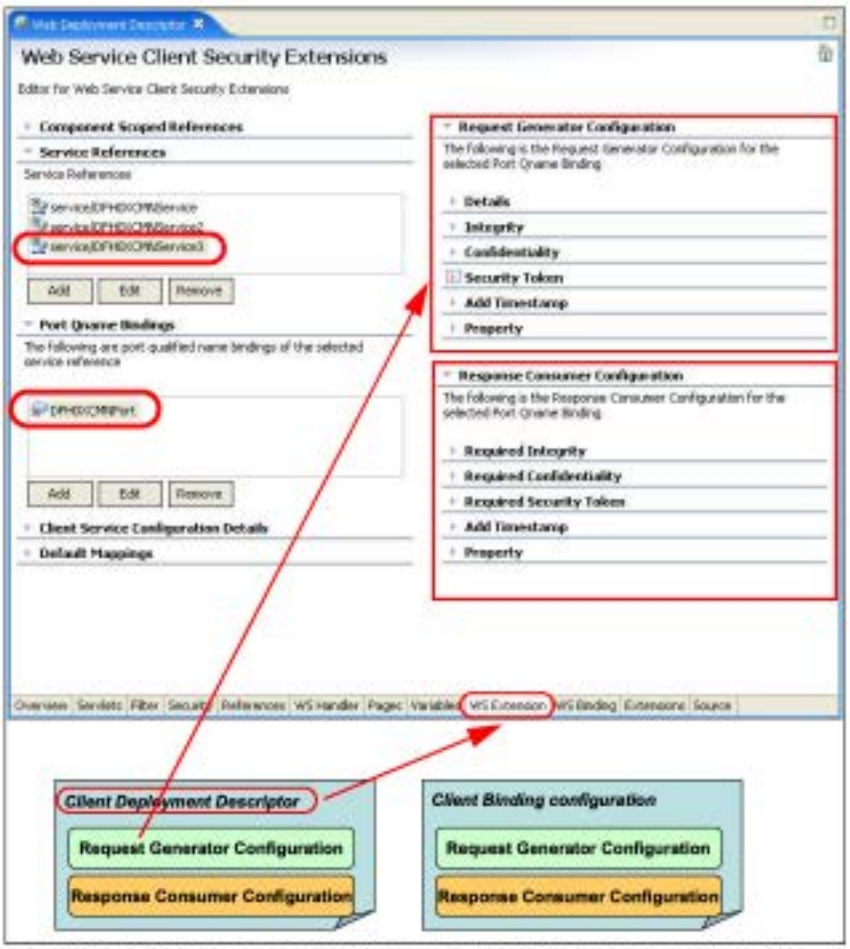


Figure 8-8 WS Extension page in Web service client Deployment Descriptor Editor

- We clicked the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPort** (the port binding for the service reference).
- In the Request Generator Configuration, we expanded **Security Token** and clicked **Add**. We entered the name `basicauth`.

  We selected the **Username** Token type from the drop-down list. When you select a Token type, the Local name is filled in automatically (Figure 8-9 on page 262). We left the URI empty.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 290/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000290



Figure 8-9 Security Token Dialog for specifying basic authentication

We clicked **OK** and a security token was created. We then saved the configuration.

After specifying the security token, a corresponding token generator must be specified in the binding configuration. The WS Binding page is for editing the client's binding file, so you can specify how to apply the required security. We clicked the **WS Binding** tab. Figure 8-10 shows the WS Binding page in the Deployment Descriptor Editor.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000291

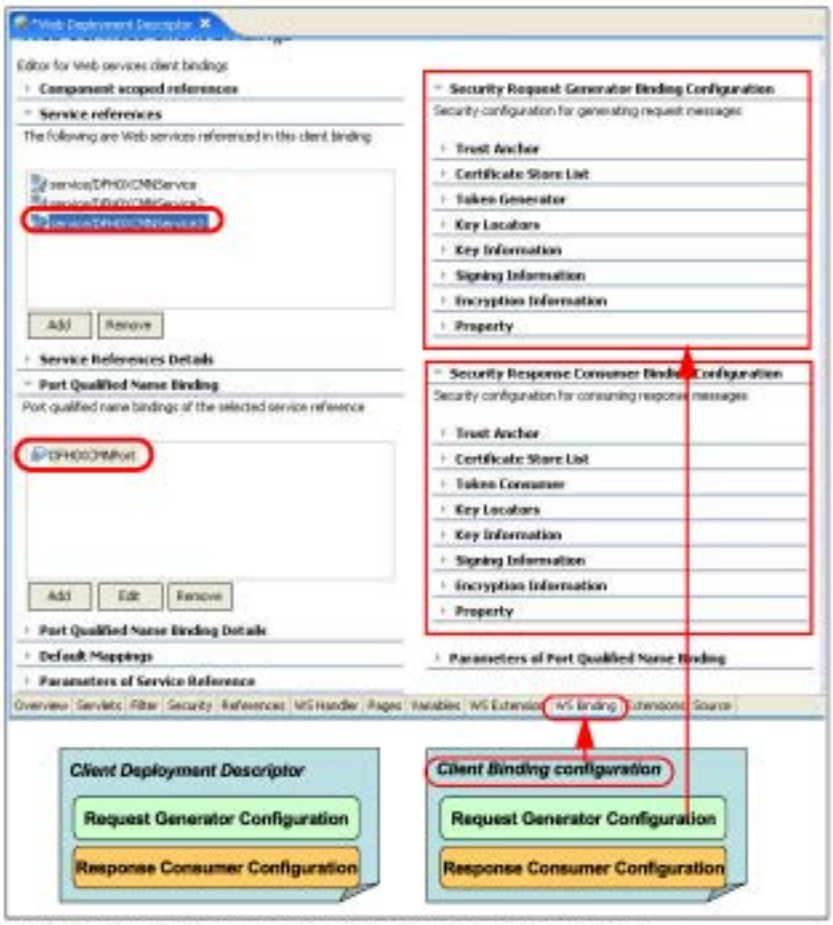


Figure 8-10 WS Binding page in the Deployment Descriptor Editor

- We selected **Token Generator** and clicked **Add**. We entered a Token generator name `basicauthToken` and allowed the Token generator class to default as `com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator`
- To select the security token, we expanded the drop-down list and selected the **basicauth** security token that we defined previously on the WS Extension page.
- We checked **Use value type** and selected the **Username Token** value type from the drop-down list. This selection filled the local name and callback handler.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 282/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000292

- We entered the user ID WEBAS3 and password REDBOOKS that we wanted to use to identify the Web service client (Figure 8-11).

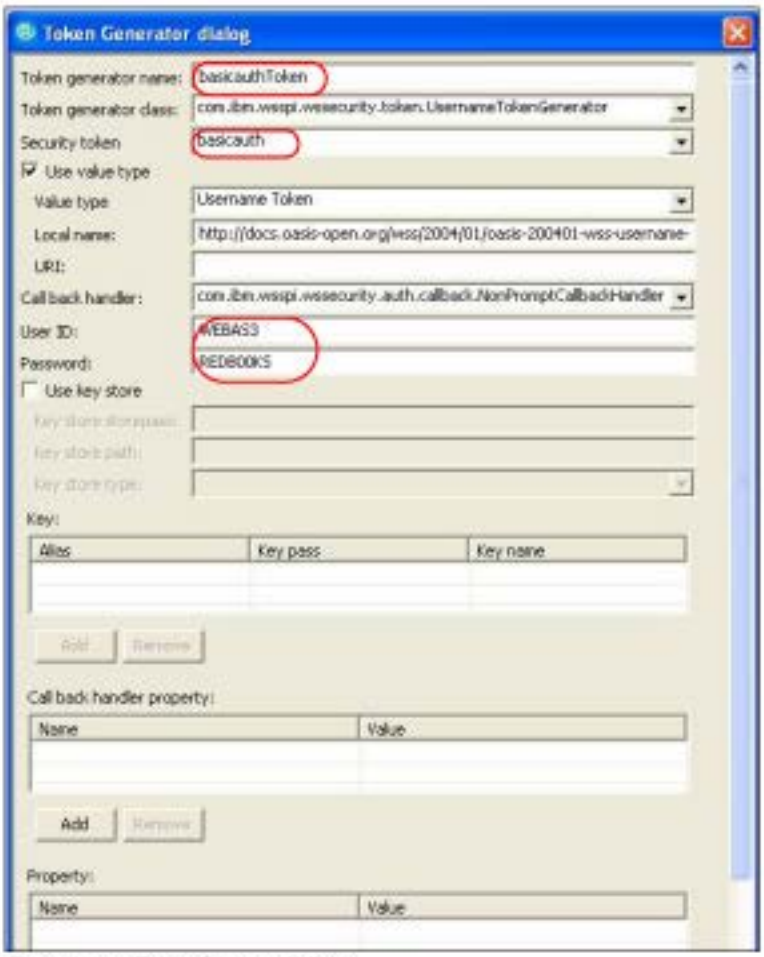


Figure 8-11 Token Generator dialog

- We clicked **OK**, and the token generator was created. We then saved the configuration.

Note: The User ID and Password in Figure 8-11 represent the credentials of the application server, *not* the browser user. The same credentials are passed to CICS in the SOAP message irrespective of the identity of the end user.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000293

Redeploying the Web service application

After configuring a security constraint for the service requester application, we exported a new EAR file called CatalogSec_WS-Security.ear. We followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec_WS-Security.ear.

8.4.2 Configuring CICS

The credentials configured using Rational Application Developer are used to form the WS-Security header that is contained in the SOAP message. An example of this message is shown in Example 8-5.

Example 8-5 SOAP security header extracted from a SOAP message

```
<wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">
    <wsse:UsernameToken>
      <wsse:Username>WEBAS3</wsse:Username>
      <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
       wss-username-token-profile-1.0#PasswordText">REDBOOK5</wsse:Password>
    </wsse:UsernameToken>
</wsse:Security>
```

This header includes the `mustUnderstand="1"` attribute, which indicates that either this header must be processed or a SOAP fault thrown.

Note: See *CICS Transaction Server for z/OS CICS Web Services Guide*, SC34-6458, for details on `mustUnderstand` and other SOAP-defined header block attributes.

In order for CICS to process the security token in the SOAP header, the pipeline configuration file must be updated to include a SOAP header processing program. Example 8-6 shows the pipeline configuration file that activates our sample security header processing program CIWSSECH.

Example 8-6 Pipeline config file, ITSO_7206_secprovider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline provider.xsd ">
  <transport>
    <default_transport_handler_list>
      <handler>
        <program>CIWSMSGH</program>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000294

```
        <handler_parameter_list/>
      </handler>
    </default_transport_handler_list>
  </transport>
  <service>
    <terminal_handler>
      <cics_soap_1.2_parser>
        <headerprogram>
          <program_name>CIWSSECH</program_name>
          <namespace>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-
                    1.0.xsd</namespace>
          <localname>Security</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.2_parser>
    </terminal_handler>
  </service>
  <appendline>CHAFLI</appendline>
</provider_pipeline>
```

The <headerprogram> element shown in Example 8-6 introduces four new elements, <program_name>, <namespace>, <localname> and <mandatory>:

- The <program_name> element contains the name of the header processing program CIWSSECH, which is to be invoked to process the security token.
- The <namespace> element is used with the <localname> element to determine which header blocks in a SOAP message should be processed by the header processing program.
- A <localname> element is also used to determine which header blocks in a SOAP message should be processed by the header processing program. The <localname> contains the element name of the header block, in our case, `Security` (see Example 8-5 on page 265).
- The <mandatory> element indicates that the header processing program must be invoked at least once, regardless of whether or not such a header has been found.

  <mandatory>true</mandatory> has the following meaning:

  During request processing in a service provider pipeline, and response processing in a service requester pipeline, the header processing program is to be invoked at least once, even if none of the headers in the SOAP message matches the <namespace> and <localname> elements:

  - If none of the headers matches, the header processing program is invoked once.
  - If any of the headers match, the header processing program is invoked once for each matching header.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000295

During request processing in a service requester pipeline, and response processing in a service provider pipeline, the header processing program is to be invoked at least once, even though the SOAP message that CICS creates has no headers initially. If you want to add headers to your message, you must ensure that at least one header processing program is invoked, by specifying <mandatory>true</mandatory> or <mandatory>1</mandatory>.

Important: If a Web service request that contains the attribute mustUnderstand="1" (or ="true") is received by CICS, and you have not coded a <headerprogram> element in your configuration file for this header, a SOAP fault message will be created by CICS.

Example 7-7 shows the fault message that CICS created when we had not specified a header processing program for the Security header.

Example 8-7 SOAP fault message created when no header program provided

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header>
    <SOAP-ENV:NotUnderstood qname="wsse:Security"
     xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecu
     rity-secext-1.0.xsd" />
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault xmlns="">
      <faultcode>SOAP-ENV:MustUnderstand</faultcode>
      <faultstring>Header block local name 'Security' is not defined to CICS.
       Mustunderstand check failed for the header block.</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The WS-Security header processing program

The header processing program CIWSSECH is invoked at every invocation of this pipeline because of the <mandatory>true</mandatory> element, but only executes entirely if the DFHFUNCTION container holds the value RECEIVE-REQUEST. Example 8-8 gives an overview of the program.

Example 8-8 Pseudo code overview of the CIWSSECH program

```
Check if invoked for RECEIVE-REQUEST, else exit
Check for correct URI, else exit
Obtain WS-Security header from DFHHEADER container
Invoke MYPARSER to parse the header
Verify the credentials extracted by MYPARSER
Put user ID into DFHWS-USERID container
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 296/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000296

Example 8-9 shows how this is done. The full program is shown in Section A.2, "Sample header processing program - CIWSSECH" on page 505.

Note: You should *not* install CIWSSECH into a production environment in its present form. It has only been coded as a example and does not contain sufficient error checking. If you wish to use it in production, you *must* review the code and make whatever changes are required for your environment.

Example 8-9 Determining if invoked to process an inbound request message

```
       GET-DFHFUNCTION.
           EXEC CICS
              GET CONTAINER('DFHFUNCTION')
              INTO(WS-FUNC-AREA)
              FLENGTH(WS-FUNC-LEN)
              RESP(WS-RESP)
           END-EXEC.

      * Copy the input container to our storage
           IF WS-FUNC-LEN NOT = 16
              MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
              PERFORM FAULT-MESSAGE
           ELSE
              IF WS-FUNC-AREA NOT = 'RECEIVE-REQUEST '
                 EXEC CICS RETURN END-EXEC
              END-IF
           END-IF.
```

Example 8-10 shows how we determined that a specific request is for the placeOrder service.

Example 8-10 Determining the requester URI

```
           PERFORM GET-DFHWS-URI.
           IF WS-URI-AREA(1:WS-URI-LEN) NOT = '/exampleApp/placeOrder'
              EXEC CICS RETURN END-EXEC
           END-IF
```

Note: The value in the DFHWS-URI container is the same value that we specified previously for the PATH attribute of the SECPORDR URIMAP in Figure 7-3 on page 192.

After verifying that we are processing a placeOrder request, the Security header (see Example 8-5 on page 265) is parsed by calling the MYPARSER program. MYPARSER is a user-written COBOL program that uses the COBOL XML PARSE statement to parse through the Security header, extracting the user ID

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000297

and password. This extracted data is then returned in a COMMAREA for the calling program to use. Example 8-11 shows the procedure division for the MYPARSER program.

Example 8-11 MYPARSER COBOL program procedure division

```
PROCEDURE DIVISION.
      MAIN-PROCESSING SECTION.
      *    Validate the commarea
           PERFORM INIT-AND-VALIDATE.
      * Received a valid COMMAREA so invoke the XML parser
      * passing it the XML message
           XML PARSE XML-DOCUMENT
              PROCESSING PROCEDURE XML-HANDLER
           END-XML.
           PERFORM RETURN-RESPONSE.
           EXEC CICS RETURN END-EXEC.
      MAIN-PROCESSING-EXIT. EXIT.
```

The XML PARSE statement parses the data in XML-DOCUMENT, using the XML-HANDLER procedure. The XML-EVENT field is evaluated and if a match is found, the code for that event is executed. The START-OF-ELEMENT and CONTENT-CHARACTERS events are handled as shown in Example 8-12

Example 8-12 Extract of code from MYPARSER program

```
      XML HANDLER SECTION.
           EVALUATE XML-EVENT
              WHEN 'START-OF-ELEMENT'
      * check if we have an element of interest
      * i.e. Username and Password
                 IF XML-TEXT = 'wsse:Username'
                    MOVE 'Y' TO IN-ELEM USERNAME-XML-TAG-FOUND
                 ELSE
                    IF XML-TEXT = 'wsse:Password'
                       move 'Y' to in-elem-flag
                    end-if
                 END-IF
              WHEN 'CONTENT-CHARACTERS'
      * If we are in an element we are interested in,
      * then extract its value
                 IF IN-ELEM = 'Y'
                    PERFORM EXTRACT-USER-ID
                 ELSE
                    IF IN-REF-REQ = 'Y'
                        PERFORM EXTRACT-PASSWORD
                    END-IF
                 END-IF
              WHEN 'END-OF-ELEMENT'
```


Document: CD1W2F3 6827 11E7 B96C 4CE39A1668C
Confidentiality: IBM



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000298

```
                CONTINUE
             WHEN 'START-OF-DOCUMENT'
                CONTINUE
```

Most of the other XML event types are handled with the CONTINUE statement because we have no need to process them. The full MYPARSER program is shown in Section A.4, "Sample XML parser program - MYPARSER" on page 522.

The parsed credentials are then verified, as shown in Example 8-13. If the user ID and password verify successfully, then the user ID is stored into the DFHWS-USERID container. This causes the business logic transaction ORDR to be executed with this user ID.

Example 8-13 Verifying the flowed credentials

```
             EXEC CICS VERIFY
                PASSWORD(CA-PASSWORD)
                USERID(CA-USER-ID)
                RESP(WS-RESP)
             END-EXEC
*       Succesful ?
             IF WS-RESP = DFHRESP(NORMAL)
                PERFORM SET-USER-ID
             ELSE
                MOVE WS-NOT-AUTH TO WS-FAULT-STRING
                PERFORM FAULT-MESSAGE
             END-IF
```

Note: In the same way that we are using the header processing program for validating security credentials, it is quite feasible to use such a program to provide some form of audit trail.

We completed the CICS setup for this scenario by permitting access to the ORDR transaction to the user ID WEBAS3:

```
PERMIT CIWS3D.ORDR CLASS(GCICSTRN) ID(WEBAS3) ACCESS(READ)
```

8.4.3 Testing SOAP message security

Now when we submit an order request on the Web browser, the credentials are added to the SOAP Security header and are extracted by the header processing program CIWSSECH. Figure 8-12 shows the ORDR transaction running with the user ID WEBAS3, while the CPIH transaction continues to execute with the CIWSNW user ID.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000299

```
STATUS:  RESULTS - OVERTYPE TO MODIFY
  Tas(0000139) Tra(CEMT) Fac(0135) Ter Pri( 255 )
    Sta(TO) Use(CICSA  ) Dsn(BCFAD2535FEF8767)
  Tas(0000459) Tra(CPIH)           Sus Tas Pri( 001 )
    Sta(L ) Use(CICSHN  ) Log(BDFAD5D4E2ED1740) Hty(0205400)
  Tas(0000460) Tra(ORDR)           Sus Tas Pri( 001 )
    Sta(L ) Use(WEBAS3  ) Log(BDFAD5D46F43FF1BF) Hty(BDF     )
  Tas(0000467) Tra(CIHF) Fac(0414) Sus Ter Pri( 001 )
    Sta(SD) Use(CICSA  ) Dsn(BCFAD5D4E52AE4BE1 Hty(2C20AAC0)

                                                    SrSID=SIC) APPLID=A6P053CL
```

Figure 8-12 CADR executing with SOAP security header user ID

8.4.4 SOAP fault messages

The <Fault> element in a SOAP message is used for reporting errors. CICS provides an application programming interface for creating, adding, or deleting SOAP fault messages. See the *CICS Transaction Server for z/OS CICS Application Programming Reference*, SC34-6434, and *CICS Transaction Server for z/OS CICS Web Services Guide*, SC34-6458, for further details on these commands.

If the credentials extracted do not validate, then CIWSSECH creates a SOAP fault message (Example 8-14) which is returned to the service requester.

Example 8-14 Generating a SOAP fault

```
           EXEC CICS VERIFY
              PASSWORD(CA-P455WORD)
              USERID(CA-USER-ID)
              RESP(WS-RESP)
           END-EXEC
      *    Successful ?
           IF WS-RESP = DFHRESP(NORMAL)
              PERFORM SET-USER-ID
           ELSE
              MOVE WS-NOT-AUTH TO WS-FAULT-STRING
              PERFORM FAULT-MESSAGE
           END-IF
         .
         .
         .
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000300

```
 FAULT-MESSAGE SECTION.
*---------------------------------------------------------------*
* Generate a SOAP Fault
*---------------------------------------------------------------*
     EXEC CICS
         GET CONTAINER('DFHWS-SOAPLEVEL')
         INTO(WS-SOAP-LEVEL)
         FLENGTH(WS-HEAD-LEN)
         RESP(WS-RESP)
     END-EXEC.

* MOVE CORRECT VERSION OF FAULTCODE
     IF WS-SOAP-11 MOVE DFHVALUE(CLIENT) to WS-FAULT-CODE
     ELSE
        MOVE DFHVALUE(SENDER) to WS-FAULT-CODE
     END-IF

     EXEC CICS SOAPFAULT CREATE
               FAULTSTRING(WS-FAULT-STRING)
               FAULTSTRLEN(length of WS-FAULT-STRING)
               FAULTCODE(WS-FAULT-CODE)
     END-EXEC.
 FAULT-MESSAGE-END. EXIT.
```

Figure 8-13 shows the fault string `Not authorized to place order`, which occurs when the user's credentials are invalid.



Figure 8-13 Browser - SOAP fault message for 'not authorized' error

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 301/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000301

If the user ID and password supplied in the SOAP Security header validate correctly, but the user ID has no access to the ORDR transaction, then a RACF error message is issued, as seen in Example 8-15.

Example 8-15 RACF messages for insufficient access

```
ICH408I USER(CIWSPC  ) GROUP(SYS1    ) NAME(PAOLO )
  CIWS3D.ORDR CL(TCICSTRN)
  INSUFFICIENT ACCESS AUTHORITY
  ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

Figure 8-14 shows the fault string that is generated by CICS for this error.



Figure 8-14 Browser - Internal server error

8.5 Enabling SSL/TLS

By default, SOAP messages are flowed in clear text. This is likely to be unacceptable in many cases. In order to address this, some form of encryption is required. SSL/TLS is a well understood and popular way of encrypting message flows. When the client connects with SSL/TLS, privacy of the data is obtained by encrypting the data.

In this section we show how we configured both the CICS and WebSphere environments to use an SSL/TLS connection (Figure 8-15).

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
Online/Notary.net
Page 302/89

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000302

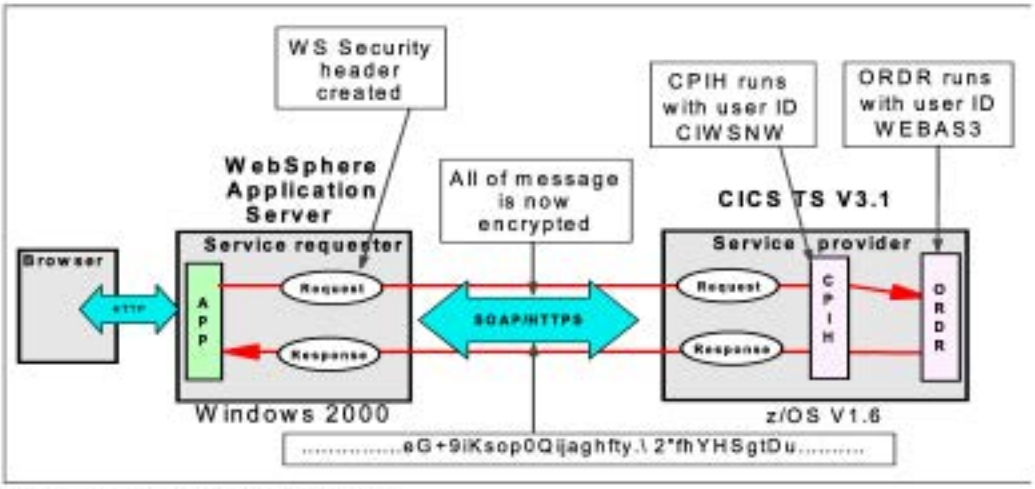


Figure 8-15 Enabling SSL/TLS scenario

We document the following steps that we followed to enable SSL/TLS:

- Creating a key ring and certificates on z/OS for CICS
- Enabling an SSL/TLS connection from WebSphere
- Configuring CICS support for SSL/TLS

8.5.1 Creating a key ring and certificates on z/OS for CICS

In order to create the certificates required for using SSL/TLS, we first needed to prepare RACF. We issued the commands shown in Example 8-16. For more information, see the chapter titled "Configuring CICS to use SSL" in *CICS Transaction Server for z/OS RACF Security Guide, SC34-6454*.

Example 8-16 RACF commands to prepare for running the DFH$RING exec

```
RDEFINE FACILITY (IRR.DIGTCERT.ADD)
RDEFINE FACILITY (IRR.DIGTCERT.CONNECT)
RDEFINE FACILITY (IRR.DIGTCERT.GENCERT)

SETROPTS RACLIST(FACILITY) REFRESH
SETROPTS GENERIC(FACILITY) REFRESH

PERMIT IRR.DIGTCERT.*       CLASS(FACILITY) ID(CIWS3D) ACCESS(READ)
PERMIT IRR.DIGTCERT.CONNECT CLASS(FACILITY) ID(CIWS3D) ACCESS(CONTROL)
PERMIT IRR.DIGTCERT.GENCERT CLASS(FACILITY) ID(CIWS3D) ACCESS(CONTROL)
```


Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineNotary.net
Page 303/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000303

The sample clist DFH$RING is provided by CICS TS to help you build a RACF KEYRING for CICS SSL use. We copied the DFH$RING clist from hlq.SDFHSAMP into our own private library and modified the clist using the customization values shown in Table 8-4.

Table 8-4 Customizing the DFH$RING clist - Site settings

Name	Value
organization	TSO
department	PSSC
certifier	CICS-Sample-Certificator
city	Montpellier
state	Herault
country	FR

These values represent the entity for which we are creating the certificates and key ring. As can be seen, these were chosen to indicate the IBM site where the Redbook residency was held.

Further parameters were supplied as input to the clist for the specific key ring to be created (Table 8-5).

Table 8-5 Customizing the DFH$RING clist - Key ring settings

Name	Value
hlqname	Cews
aslname	Cewss3c1
ipname	mvsg3.mop.ibm.com
userid	CIWS3D

The name of the key ring to be created is Cews.Cewss3c1.KeyRing. The userid parameter was supplied as CIWS3D (the CICS region user ID). The ipname is the address of the z/OS on which our CICS region CIWSS3C1 runs.

Further modifications were made to the clist because we chose not to generate certificates for a Web server or for an EJB container. Instead we generated certificates for a Web service Client, Web service Server, and a default certificate. These changes are shown in Example 8-17.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000304

Example 8-17 Customizing the DFHSTRNG EXEC - certificate labels

```
'RACDCERT ID('faruser') DELETE(LABEL('lastname' Web service Client'))'
'RACDCERT ID('faruser') DELETE(LABEL('lastname' Web service Server'))'
'RACDCERT ID('faruser') DELETE(LABEL('lastname' Default Certificate'))'
```

We used the batch job shown in Example 8-18 to execute the DFHSRING clist in order to create the required RACF profiles

Example 8-18 Batch job to invoke the DFHSRING clist

```
//RUNJOB  JOB 1,CICS,TIME=1440,NOTIFY=&SYSUID,REGION=4M,
//        CLASS=A,MSGCLASS=X,MSGLEVEL=(1,1)
//*
//RUN     EXEC PGM=IKJEFT01
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
 %DFHSRING CICS CICSSAC1 mvsg1.mop.ibm.com HOSTSERV(CICSSSL)
/*
```

To complete this part of the configuration we exported the Certificate Authority (CA) certificate in Base64 X.509 format using the RACF RACDCERT command:

```
RACDCERT CERTAUTH EXPORT(LABEL('CICS-Sample-Certification')) +
DSN(CICSSG4.CERTAUTH.X509) FORMAT(CERTB64)
```

8.5.2 Enabling an SSL/TLS connection from WebSphere

To enable an SSL/TLS connection from WebSphere we did the following:

- FTPed the certificate from z/OS to our client machine
- Imported the certificate into the WebSphere Application Server keystore
- Enabled the service requester to use HTTPS

FTP the certificate from z/OS

We used FTP as shown in Example 8-19 to send the exported certificate in ASCII format to our client machine.

Example 8-19 Using FTP to send an exported CA certificate to a client workstation

```
C:\>ftp mvsg3.mop.ibm.com
Connected to mvsg3.mop.ibm.com.
220-FTPD1 IBM FTP CS V1R6 at MVSG3.pssc.mop.ibm.com, 13:23:13 on 2005-11-30.
220 Connection will close if idle for more than 5 minutes.
User (mvsg3.mop.ibm.com:(none)): ciwsga
```


Document Id: CD1W2F3 6827 11E7 B96C 4CE29A1 6BEC
Confidentiality: IBM



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000305

```
331 Send password please.
Password:
230 CIWSGA is logged on.  Working directory is "CIWSGA.".
ftp> ascii
200 Representation type is Ascii NonPrint
ftp> get CERTAUTH.X509
200 Port request OK.
125 Sending data set CIWSGA.CERTAUTH.X509
250 Transfer completed successfully.
ftp: 1022 bytes received in 0.00Seconds 1022000.00Kbytes/sec.
ftp> quit
221 Quit command received. Goodbye.
```

Importing the certificate

Next, we needed to make the CERTAUTH.X509 certificate available for use by WebSphere Application Server. To do this, we imported the CERTAUTH.X509 certificate to the WebSphere Application Server keystore (DummyServerKeyFile.jks) using iKeyMan.

Note: The iKeyMan tool is provided by the IBM Java SDK. On our workstation the tool was located in directory C:\Program Files\IBM\Java142\jre\bin.

After opening a command prompt window we set the PATH variable and typed ikeyman to invoke the iKeyMan GUI.

```
set PATH=C:\Program Files\IBM\Java142\jre\bin\;%PATH%
ikeyman
```

We used iKeyMan to add the CERTAUTH.X509 file to the WebSphere Application Server keystore (ServerKeyFile.jks) as shown in Figure 8-16.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000306

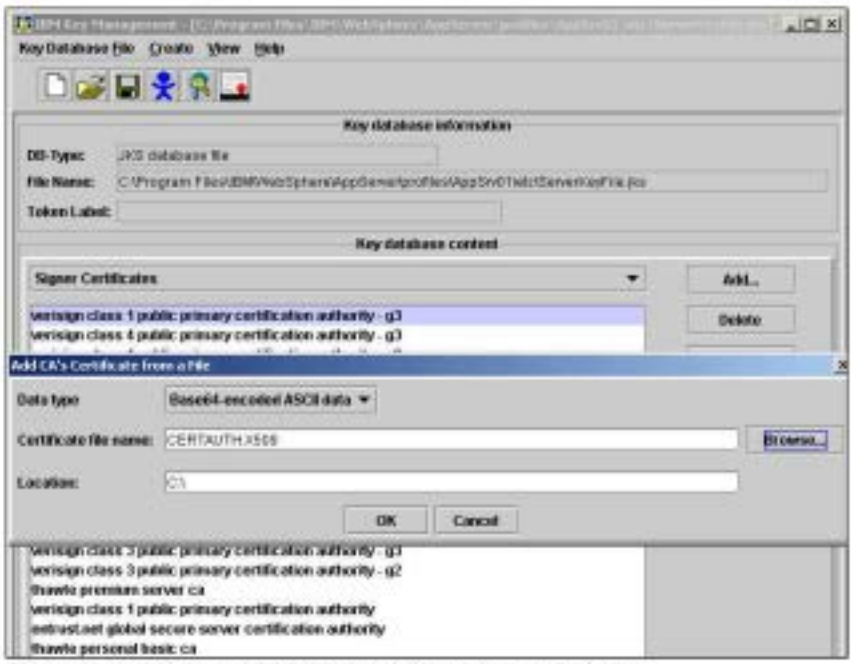


Figure 8-16 IKeyman - Add CICS CA certificate from an X509 file

After we clicked **OK** to proceed with this action, we were prompted to enter a label for this certificate. Figure 8-17 shows the CICS CA certificate added to the WebSphere keystore.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 307/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000307

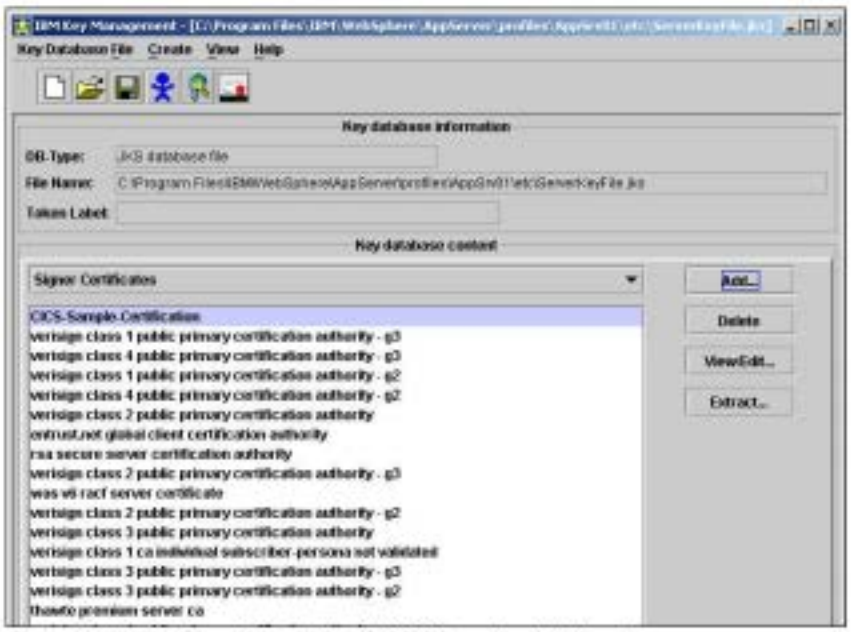


Figure 8-17 IKeyMan - After adding CICS CA certificate to keystore

Next we created a new JSSE repertoire by logging onto the admin console and clicking **Security → SSL → New JSSE repertoire**. We entered the Alias name as `CICS-SSLSettings` and defaulted the Protocol to SSL_TLS. The Key file and Trust file names were both specified as `c:\Program Files\IBM\WebSphere \Appserver\profiles\AppSrv01\bin\ServerKeyFile.jks` and the password specified was `WebAS`. Changed fields are shown in Table 8-6, all other values were allowed to default.

Table 8-6 Altered values for new JSSE repertoire

Field name	Value
Alias	CICS-SSLSettings
Key file name	C:\Program Files\IBM\WebSphere\AppServer\profiles\ AppSrv01\bin\ServerKeyFile.jks
Key file password	WebAS
Trust file name	C:\Program Files\IBM\WebSphere\AppServer\profiles\ AppSrv01\bin\ServerKeyFile.jks
Trust file password	WebAS

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 308/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000308

We clicked **OK** → **Save** → **Save** to save the configuration. Next we stopped and restarted the application server, in order to have the new repertoire recognized.

Enabling the service requester to use HTTPS

In Section 8.4.1, "Configuring the service requester" on page 258 we showed that to configure our Web service client to use WS-Security, we used Rational Application Developer V6.0. We next used a similar process to enable the service requester to use HTTPS:

1. We imported the client application archive CatalogSec_WS-Security.ear into RAD. We then expanded the Dynamic Web Project (`CatalogSecWeb`) in the Project Explorer and opened (double-clicked) the deployment descriptor.
2. After the Deployment Descriptor Editor opened, we selected the **WS Binding** page. The WS Binding page is for editing the client's binding file, so you can specify how to apply the required security.
3. We clicked the service reference **service/DFH0XCMNService** and then the Port Qualified Name Binding **DFH0XCMNPort.**
4. In the Port Qualified Name Binding Details section we specified our SSL repertoire name `Alp3-ITSO1Node01/CICS-SSLSettings` for the name of the HTTP SSL Configuration (Figure 8-18).



Figure 8-18 RAD - Enabling HTTPS for service requester

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000309

5. We repeated steps 3 and 4 for our other CICS Web services `service/DFHOXCMNService2` and **`service/DFHOXCMNService3`**.
6. We then saved the configuration and exported a new EAR file called CatalogSec_WS-Security_HTTPS.ear.

Note: The fully qualified name, including the node name, must be specified for HTTP SSL configuration: Name in order for this to be recognized by WebSphere Application Server when the application is deployed.

After enabling the service requester to use HTTPS, we needed to re-deploy the application using the WebSphere admin console. We followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec_WS-Security_HTTPS.ear.

See "Testing SSL/TLS" on page 282 for information on how we tested SSL/TLS connectivity from WebSphere Application Server to CICS.

8.5.3 Configuring CICS support for SSL/TLS

To activate SSL/TLS support in our CICS TS V3.1 region, we specified the following system initialization parameter:

- `KEYRING=Ciws.Ciwss3cl`

  This specifies the name of a key ring we created in the RACF database that contains keys and certificates used by this CICS region.

We allowed the ENCRYPTION, MAXSSLTCBS, SSLCACHE and SSLDELAY system initialization parameters to default. We then restarted the CICS region.

Tip: You should restart your CICS region after updating the system initialization parameter and before creating the TCPIPSERVICE for SSL. The CIPHERS attribute of the TCPIPSERVICE resource is only automatically updated by CICS if CICS has been started with SSL support active.

Changing the TCPIPSERVICE

To enable SSL connections in CICS we created a new TCPIPSERVICE with the PORTNUMBER, CERTIFICATE and SSL attributes set as shown in Figure 8-19.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000310

```
 OVERTYPE TO MODIFY                                        CICS RELEASE = 0640
  CEDA  Alter TCpipservice( SSCISSL  )
   TCpipservice   : SSCISSL
   GROup          : SSCJ
   DEscription    ==> TCPIPSERVICE
   Urm            ==> DFHWBADX
   POrtnumber     ==> 14303              1-65535
   STatus         ==> Open               Open | Closed
   PROtocol       ==> Http               Iiop | Http | Eci  User
   TRansaction    ==> CWXN
   BAcklog        ==> 00005              0 27hr
   TSqprefix      ==>
   Ipaddress      ==>
   SOcketclose    ==> No                 No | 0-240000 (HHMMSS)
   MAxdatalen     ==> 000032             3-524288
  SECURITY
   SSl            ==> Yes                Yes | No | Clientauth
   CErtificate    ==> Class3el-Web-service-Server
 +  (Mixed Case)

                                                          SYSID SSCI APPLID WMPSSCI
```

Figure 8-19 TCPIPSERVICE Updated for SSL support

We noticed that CICS automatically initialized the CIPHERS attribute with a default list of acceptable codes (Example 8-20).

Example 8-20 TCPIPSERVICE CIPHERS attribute

```
CIphers      ==> 0504353E373839F3031323330A1613100009152F3F0C03060201
```

8.5.4 Testing SSL/TLS

To test the SSL/TLS scenario, we needed to configure the catalog manager J2EE application so that it accesses the catalog services using HTTPS. We started a Web browser session and entered the URL:

```
http://cam21-pc3:9080/CatalogWeb/Welcome.jsp
```

We clicked **CONFIGURE** and specified the encrypted addresses of the Catalog services (Figure 8-20). We clicked **SUBMIT** to complete the update.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000311

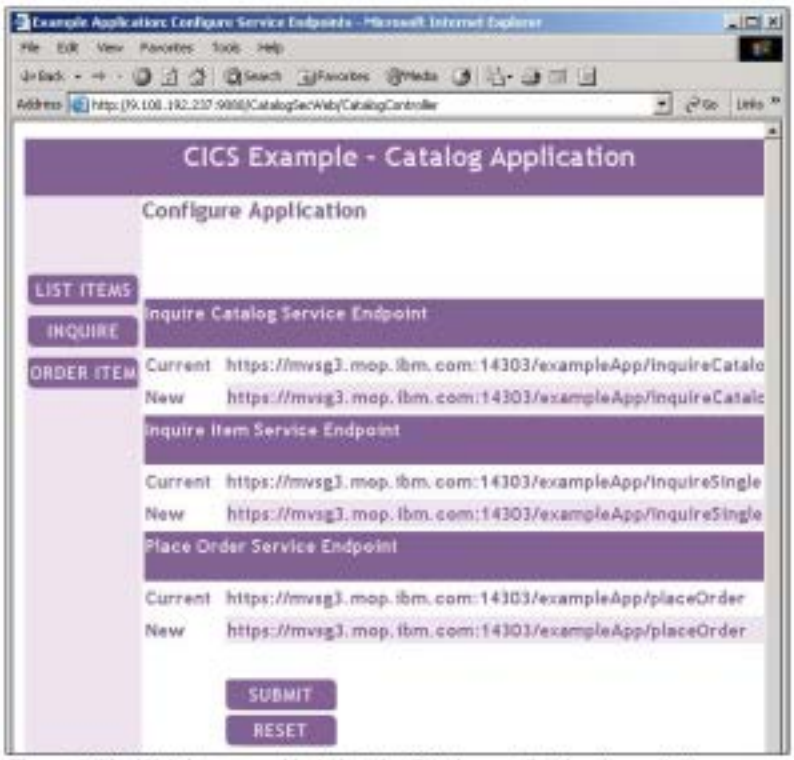


Figure 8-20 Web browser - Updating the Catalog application to use https

We then tested each of the three services (inquireCatalog, inquireSingle, and placeOrder) to ensure that all worked correctly. A display of TCP/IP connections on z/OS showed a connection to CICS region CIWSS3C1 on port 14303 (Example 8-21).

Example 8-21 Display of IP connections

```
Display  Filter  View  Print  Options  Help
-------------------------------------------------------------------------------
SDSF ULOG  CONSOLE CIWSGA                          LINE 42      COLUMNS 42- 121
COMMAND INPUT ===>                                             SCROLL ===> CSR
 -D TCPIP,,NETSTAT,CONN
  EZZ2500I NETSTAT CS V1R6 TCPIP 328
  USER ID  CONN     LOCAL SOCKET           FOREIGN SOCKET         STATE
  BPXOINIT 00000011 0.0.0.0..10007         0.0.0.0..0             LISTEN
  CIWSR3C1 00008C23 0.0.0.0..13301         0.0.0.0..0             LISTEN
  CIWSR3C1 00008C87 9.100.193.167..13301   9.100.199.156..4338    ESTBLSH
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000312

```
CICSS3C2 00000025 0.0.0.0..13332        0.0.0.0..0          LISTEN
CICSS3D1 00000B96 9.100.193.167..14303  9.100.192.237..1086 ESTBLSH
CICSS3C1 00000B71 0.0.0.0  14333        0.0.0.0  0          LISTEN
```

A CEMT INQ TCPIPSERVICE command shows that this port has the SSL attribute and that one connection has been established (Figure 8-21).

```
 INQUIRE TCPIPSERVICE
  STATUS:  RESULTS - OVERTYPE TO MODIFY
   Tcpips(SECSSL  ) Ope Por(14302) Http Ssl Tra(CWXN)
     Con(00001) Bac( 00000 ) Max( 000032 ) Urm( CICSWEB2 ) Soc
                                                        SYSID=5EC1 APPLID=AGPCS3C1
```

Figure 8-21 TCPIPSERVICE - SSL support installed and active

8.6 Enabling SOAP message security with WMQ

In this scenario, we show how SOAP Security headers can be used with WMQ. We customize the catalog application so that when the ORDR transaction executes, it calls a Web service to dispatch the order. This Web service is located in a separate CICS region and the transport mechanism used is WMQ. To make the dispatchOrder service (CICS transaction DISP) run with the same user ID as the calling transaction (ORDR), we transfer the caller's identity in a SOAP Security header.

This scenario was chosen for two reasons.

- To demonstrate the use of a SOAP Security header in a CICS service request
- To demonstrate the use of SOAP Security headers with WMQ

Figure 8-22 shows how we configured the environment.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000313

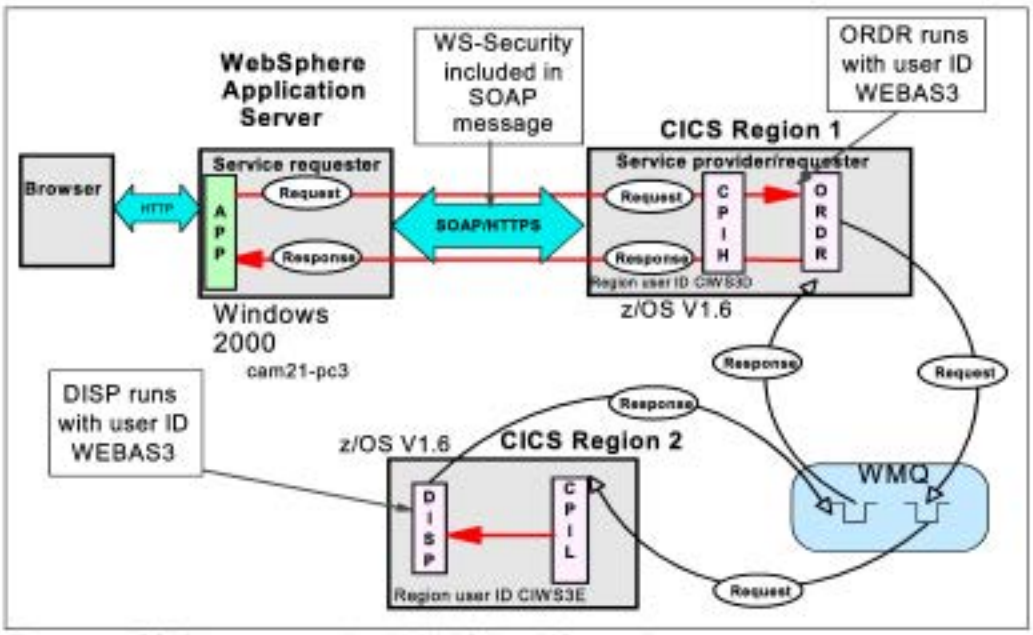


Figure 8-22 SOAP message security with WebSphere MQ scenario

Note: This scenario is an example of *identity assertion*, in which an intermediary server (CICS region 1) authenticates the client and transfers the request message and identity to the target server (CICS region 2). See "Identity assertion" on page 244 for more information on the concept of identity assertion.

We document the following steps necessary to set up this security scenario:

- Configuring CICS to use WMQ
- Configuring the service requester
- Configuring the service provider
- Configuring security for WMQ

8.6.1 Configuring CICS to use WMQ

For this, we rely heavily upon the work already done and described in Chapter 4, "Web services using WebSphere MQ" on page 111. We note only where differences occur, otherwise we used the same values.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineLibrary.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000314

We added WebSphere MQ support to our two CICS regions CIWSS3C1 and CIWSS3C2. For the dispatchOrder service, the service provider is CIWSS3C2 and the service requester is CIWSS3C1. Table 8-7 shows the different settings, parameters, and values used for these two CICS regions.

Table 8-7 Different settings for the two CICS regions

	CICS Region 1	CICS Region 2
WMQ queue manager	MQS3	MQS3
INITQ	WSB3.S3C1.INITQ	WSB3.S3C2.INITQ
Pipeline	PIPE2	PIPE3
Configuration file	/CIWS/S3C1/config/ITSO.7206.secrequester.xml	/CIWS/S3C2/config/ITSO.7206.secprovider.dispatch.xml
HFS structure	/CIWS/S3C1/...	/CIWS/S3C2/...
Header processing program	CIWSSEC1	CIWSSEC3
Region user ID	CIWS33	CIWS3E
Jobname	CIWSS3C1	CIWSS3C2
Applid	A6PCS3C1	A6PCS3C2

The WebSphere MQ queuenames used were WSB3.S3C2.PIPE3.REQUEST and WSB3.S3C2.PIPE3.RESPONSE while the process name was WSB3.S3C2.PROCESS.

The security of the CICS transactions supplied by WebSphere MQ resources was protected by issuing the RACF commands shown in Example 8-22.

Example 8-22 RACF commands to protect WebSphere MQ CICS transactions

```
RDEFINE GCICSTRN WS3MQUSR UACC(NONE) +
         ADDMEM(CIWS3E.CKBM, CIWS3E.CKCN, CIWS3E.CKDL, -
                CIWS3E.CKCP, CIWS3E.CKQC, CIWS3E.CKRE, -
                CIWS3E.CKRT, CIWS3E.CKSD, CIWS3E.CKSQ, -
                CIWS3E.CKPC, CIWS3E.CKTB, CIWS3E.CKRC, -
                CIWS3E.CKHQ, CIWS3E.CKSS, CIWS3E.CKSA) -
         OWNER(IBMUSER)

PERMIT WS3MQUSR CLASS(GCICSTRN) ID(STL)
RALTER GCICSTRN CIWS3E.ADDMEM(CIWS3E.CKTL, CIWS3E.CKAM) OWNER(IBMUSER)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

In Example 8-22 we:

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000315

- Define the transaction group WS3MQUSR and add transactions CKBM, CKCN, CKDL, CKDP, CKQC, CKRS, CKRT, CKSD, CKSQ, CKMC, CKMH, CKRC, CKRQ, CKSG and CKSV to the group.
- Permit access to the transaction group WS3MQUSR to the RACF user group SYS1 (SYS1 is the group of administration user IDs on our system)
- Alter the transaction group CIWS3T (used for CICS region CIWSS3C2) by adding transactions CKTI and CKAM to the group.
- Refresh the RACF CICS transaction class TCICSTRN.

These commands were repeated with the prefix of CIWS3D for the CIWSS3C1 CICS region

8.6.2 Configuring the service requester

In this scenario the service requester is the ORDR transaction itself. This demonstrates that a transaction can act as both a service provider and a service requester.

Configuring the sample application

We configured the catalog application in the same way as in "Configuring the Catalog application" on page 122, except that we specified the Outbound WebService URI as:

```
jms:/queue?destination=VSG3.S3C2.PIPE3.REQUEST2NQS3&targetService=/e
xampleApp/dispatchOrder&replyDestination=VSG3.S3C2.PIPE3.RESPONSE
```

This means that the SOAP request for the dispathOrder service is sent using WMQ, via the WMQ queue VSG3.S3C2.PIPE3.REQUEST, to an application servicing that queue. The response is returned to the ORDR transaction via the queue VSG3.S3C2.PIPE3.RESPONSE.

8.6.3 Header processing program

In order to add a Security header to the orderDispatch SOAP message, we wrote a new header processing program CIWSSECR that inserts a <wsse:Security> header into the message. A synopsis of this program is shown in Example 8-23.

Example 8-23 Header processing program to insert the SOAP security header

```
Check if invoked for SEND-REQUEST, else exit
Check for correct URI, else exit
Obtain user ID from DFHWS-USERID container and insert into Security header
Put Security header to DFHHEADER container
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000316

The full CIWSSECR program is shown in A.5 "Sample header processing program - CIWSSECR" on page 527.

Pipeline configuration file

A new requester pipeline configuration file was created by copying the CICS-supplied sample from basicsoap11requester.xml to /CIWS/S3C2/config/ITSO_7206_secrequester.xml. The header processing program was then added to the configuration file as shown in Example 8-24. The <program_name> element contains the name of the header processing program CIWSSECR, which is to be invoked to create the security token.

Example 8-24 Configuration file for pipeline PIPE2

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/
                    pipeline requester.xsd ">
 <service>
  <service_handler_list>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>CIWSSECR</program_name>
          <namespace>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-ws
           security-secext-1.0.xsd</namespace>
          <localname>Security</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
  </service_handler_list>
 </service>
</requester_pipeline>
```

In CICS, the new PIPELINE resource definition was created by copying PIPE1 as PIPE2 and modifying the CONFIGFILE attribute to the name of the new pipeline configuration file.

8.6.4 Configuring the service provider

A new header processing program CIWSSECS and pipeline configuration file were required.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000317

Header processing program

Program CIWSSECS is similar to CIWSSECH except that no password verification is done because no password has been sent. A short description of this program is shown in Example 8-25.

Example 8-25 Header processing program logic to change user ID

```
Check if invoked for RECEIVE <SOAP>, else exit
Check for correct URI, else exit
Obtain Security header from DFHHEADER container
Parse the header and obtain user ID
Put user ID into DFHWS-USERID container
```

Pipeline configuration file

The new configuration file /CIWS/S3C2/config/ITSO_7206_secprovider_c spatch.xml, was created by copying /CIWS/S3C1/config/ITSO_7206_secprovider.xml (see Example 8-6 on page 265). The C WSSECS program was specified in place of CIWSSECH.

Example 8-26 Configuration file for pipeline PIPE3

```
<?xml version="1.0" encoding="UTF-8"?>
<provider pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <transport>
      <default_transport_handler_list>
          <handler>
             <program>CIWSMSG</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
  </transport>
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>CIWSSECS</program_name>
          <namespace>http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-ws
          security-secext-1.0.xsd</namespace>
          <localname>Security</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
```


Document is CONFIDENTIAL per IBM Agreement
Do not copy

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000318

```
  </service>
  <appendler>DFHPITP</appHandler>
</provider pipeline>
```

8.6.5 Configuring WebSphere MQ for security

Since in this scenario we are passing a user ID from CICS region 1 to CICS region 2 without re-authenticating, it is necessary to establish a trust relationship between the CICS regions. We can do this by restricting access to the specific WMQ queues used to transport the dispatchOrder request.

We secured the WMQ connection between CICS region 1 and CICS region 2 as follows:

- Configure general security for WMQ

  The RACF commands that we issued in order to configure general security for WMQ are listed in Example 8-27.

  Example 8-27 RACF commands to configure WebSphere MQ security

```
SETROPTS CLASSACT(MQADMIN,MQQUEUE,MQPROC,MQNLIST,MQCONN,MQCMDS)
SETROPTS  GENERIC(MQADMIN,MQQUEUE,MQPROC,MQNLIST,MQCONN,MQCMDS)
RDEFINE MQADMIN MQS3.NO.CMD.CHECKS
RDEFINE MQADMIN MQS3.NO.CMD.RESC.CHECKS
RDEFINE MQADMIN MQS3.NO.PROCESS.CHECKS
RDEFINE MQADMIN MQS3.NO.NLIST.CHECKS
RDEFINE MQADMIN MQS3.NO.CONTEXT.CHECKS
RDEFINE MQADMIN MQS3.NO.ALTERNATE.USER.CHECKS
RDEFINE MQADMIN MQS3.YES.SUBSYS.SECURITY
RDEFINE MQCONN  MQS3.CICS UACC(NONE)
PERMIT MQS3.CICS CLASS(MQCONN) ID(CICSSC,CICS3E) ACCESS(READ)
SETROPTS RACLIST(MQQUEUE) REFRESH
SETROPTS GENERIC(MQQUEUE) REFRESH
SETROPTS RACLIST(MQCONN) REFRESH
SETROPTS GENERIC(MQCONN) REFRESH
SETROPTS RACLIST(MQADMIN) REFRESH
SETROPTS GENERIC(MQADMIN) REFRESH
```

  The SETROPTS CLASSACT and SETROPTS GENERIC commands activate the specific and generic WMQ security classes. All classes were activated.

- Turn security on for the queue manager.

  We set the switch cmgr-name YES SUBSYS SECURITY to *Y* for our queue manager MQS3 (see "WebSphere MQ transport" on page 224).

Document ID: SG24-6827-01 ITSO Draft as of 2004-06-07
Confidential: No



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000319

- Grant access to the WMQ queues

  We granted both CICS region user IDs connect access to the queue manager and then refreshed the RACF profiles. Once those definitions were active, we then secured access to the queues, using the commands shown in Example 8-28.

Example 8-28 RACF commands to secure WebSphere MQ queues

```
RDEFINE GMQQUEUE ORDR.DISPATCH UACC(NONE) +
       ADDMEM('MQS3.VSG3.S3C2.PIPE3.REQUEST, +
              MQS3.VSG3.S3C2.PIPE3.RESPONSE')
PERMIT ORDR.DISPATCH CLASS(GMQQUEUE) ID(CICS3D,CICS3E) ACCESS(UPDATE)
RDEFINE GMQQUEUE S3C2.INITQ UACC(NONE) +
       ADDMEM('MQS3.VSG3.S3C2.INITQ')
RDEFINE GMQQUEUE S3C2.INITQ UACC(NONE) +
       ADDMEM('MQS3.VSG3.S3C2.INITQ')
PERMIT S3C2.INITQ    CLASS(GMQQUEUE) ID(CICS3D) ACCESS(UPDATE)
PERMIT S3C2.INITQ    CLASS(GMQQUEUE) ID(CICS3E) ACCESS(UPDATE)
PERMIT ORDR.DISPATCH CLASS(GMQQUEUE) ID(CICS3D) ACCESS(UPDATE)
PERMIT ORDR.DISPATCH CLASS(GMQQUEUE) ID(CICS3E) ACCESS(UPDATE)
PERMIT ORDR.DISPATCH CLASS(GMQQUEUE) ID(WEBAS3) ACCESS(UPDATE)
```

In Example 8-28 we add the response and request queues to the generic group ORDR.DISPATCH. By default, when CICS makes an API resource security check on a CICS connection, it checks to see if two user IDs have access to the resource. The first user ID checked is the CICS region user ID. The second user ID checked is the user ID associated with the CICS transaction. The CICS region user IDs (CICS3D and CICS3E) are permitted UPDATE access to the group. The user ID WEBAS3 is also granted UPDATE access.

For full details on securing access to WMQ queues, see *WebSphere MQ for z/OS System Setup Guide V5.3*, SC34-6583.

8.6.6 Testing security with WMQ

To test the WMQ security scenario, we started a Web browser session and entered the URL:

```
http://car21-pc3:9080/CatalogWeb/Welcome.jsp
```

Figure 8-23 shows the result of the initial ORDER ITEM request.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000320



Figure 8-23 Browser - INVREQ calling dispatch server

We noticed the following RACF message in the CICS log of region CIWSS3C2 (Example 8-29).

Example 8-29 RACF message when WEBAS3 has no access to DISP

```
ICH408I USER(WEBAS3  ) GROUP(CIWS    ) NAME(WEB USER1
   CIWS3E.DISP CL(TCICSTRN)
   INSUFFICIENT ACCESS AUTHORITY
   ACCESS INTENT(READ   )  ACCESS ALLOWED(NONE   )
```

This error shows that user ID WEBAS does not have access to the DISP transaction. We permitted access to the DISP transaction to the user ID WEBAS3 using the RACF commands shown in Example 8-30.

Example 8-30 RACF commands to define DISP and grant access to it

```
RDEFINE GCICSTRN WS3DISP ADDMEM(CIWS3E.DISP) UACC(NONE) OWNER(IBMUSER)
PERMIT WS3DISP CLASS(GCICSTRN) ID(WEBAS3) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
RDEFINE SURROGAT WEBAS3.DFHSTART UACC(NONE) OWNER(WEBAS3)
PERMIT WEBAS3.DFHSTART CLASS(SURROGAT) ID(CIWS3E) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

We then repeated the ORDER ITEM request successfully. Figure 8-24 on page 293 shows the DISP transaction executing with the user ID of WEBAS3 on CIWSS3C2, with CPIQ and CPIL executing with the user ID of CIWS3E.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 321/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000321

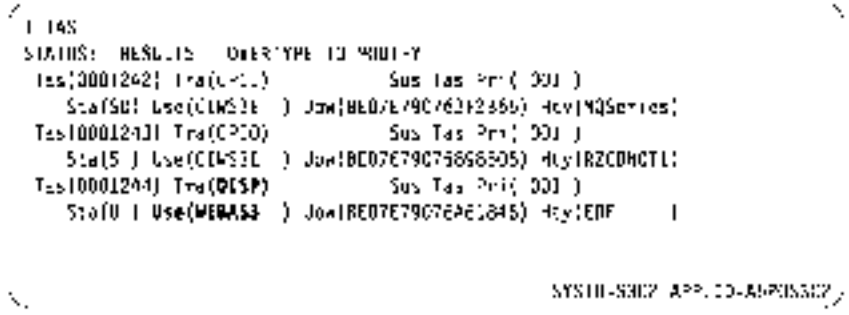


Figure 8-24 DISP transaction running in CICSS3C2

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000322

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000323

9

Security scenarios using CICS WS-Security support

In this chapter we outline several security scenarios that demonstrate how you can secure CICS Web services using the CICS-supplied message handler, DFHWSSE1.

We provide step-by-step security configuration for a number of scenarios, including:

- Basic authentication
- Signing a SOAP message
- Encrypting a SOAP message

Note: The security scenarios documented in this chapter use the CICS-supplied WS-Security support. See Chapter 8, "Security scenarios" on page 247 for WS-Security scenarios using custom message handlers.

We show how we configured both CICS and WebSphere Application Server for these security scenarios.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000324

9.1 Preparation

Security techniques for securing Web services were discussed in Chapter 7, "Securing Web services" on page 213. In this chapter, we describe the following security scenarios:

- Basic authentication

  In this scenario we show how the CICS-supplied message handler DFHWSSE1 can extract a WS-Security UsernameToken from a SOAP header, validate the username and password, and set the user ID of the CICS task to the username passed in the header. This scenario is described in "Basic authentication" on page 302.

- Signing a SOAP message

  In this scenario we show how CICS and WebSphere Application Server can be configured to exchange signed SOAP messages. The SOAP message sent by WebSphere Application Server contains an X.509 certificate that allows the CICS transaction to run under a user ID associated with the X.509 certificate. See "Signing a SOAP message" on page 330 for details of this scenario.

- Encrypting a SOAP message

  In this scenario we show how CICS and WebSphere Application Server can be configured to exchange encrypted SOAP messages. The SOAP message contains an encrypted body so that the content of the message cannot be understood by an unauthorized party. This scenario is described in 9.6, "Encrypting a SOAP message" on page 357.

9.1.1 Software checklist

The software we used is listed in Table 9-1.

Table 9-1 Software used in the WS-Security security scenarios

Windows	z/OS
Internet Explorer V6.0	z/OS V1.7
Windows XP V5.1 SP2	CICS Transaction Server V3.1 (with PTFs UK15271 and UK15261 installed)
IBM WebSphere Application Server - ND V6.1.0.0	RACF V1.7

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000325

Windows	z/OS
Our J2EE applications ► CatalogSec2.ear Catalog manager service requester application with no security enabled ► CatalogSec2_WS-Security_BasicAuth.ear Catalog manager service requester application with WS-security basic authentication enabled ► CatalogSec2_WS-Security_Signature.ear Catalog manager service requester application with WS-security signature enabled ► CatalogSec2_WS-Security_Encryption.ear Catalog manager service requester application with WS-security encryption enabled	Our user-written CICS programs ► CIWSMSGH (message handler program) This program changes the transaction ID for the Web service requests received in the pipeline. ► SNIFFER (message handler program) This program browses through the containers available in the pipeline

For this chapter we use a slightly different version of the Catalog application than the version used in Chapter 8. In the wsdl created by CICS the names of the Web service binding ports are the same for each of the three Web services. We noticed that this caused some problems in AST when editing the WebSphere WS-Security configuration files. We changed the name of the binding ports in the corresponding wsdl files from the default DFH0XCMNPort to DFH0XCMNPortPO for the placeOrder service, DFH0XCMNPortIS for inquireSingle and DFH0XCMNPortIC for inquireCatalog. We then recreated the Web services using our development tool (Rational Application Developer).

9.1.2 Definition checklist

The definitions we used are listed in Table 9-2

Table 9-2 Settings used in the WS-Security security scenarios

Value	CICS TS	WebSphere Application Server
IP name	wtsc.tss.ibm.com	mikee31
IP address	9.12.4.75	9.12.4.217
TCP/IP port	14301	9081
Jobname	CIWSS3C1	
APPLID	MQPOS3C1	
TCPIPSERVICE	S3C1	

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000326

Value	CICS TS	WebSphere Application Server
Provider PIPELINEs	PIPE1 used for inquireSingle and inquireCatalog services PIPEWSSE used for placeOrder service	
Configuration files	ITSO_7206_basicsoap12provider.xml ITSO_7206_wssec_basicauth_provider.xml ITSO_7206_wssec_signature_provider.xml ITSO_7206_wssec_encryption_provider.xml	
URIMAP	SECPORDR SECICATA SECISING	

The user IDs we used in our configuration are listed in Table 9-3.

Table 9-3 User IDs

Value	CICS TS
CICS region user ID	CIWS3D
CICS default user ID	CICSUSER
User IDs for which we wish to permit access	CIWSNW WEBAS1

Because the Catalog application inquireSingle and inquireCatalog services are read-only services, whereas the placeOrder service updates the database, we show here how to secure the placeOrder service. We run the inquireSingle and inquireCatalog services under a pre-defined user D (CIWSNW). The placeOrder service is run under the caller's user ID (WEBAS1).

Tip: For the examples described in this book, we permit access for single user IDs. In a production environment, you will probably create a group of users requiring common access. Once a group is built, you can permit access for the group. This allows users' access to be controlled by the group to which they belong, rather than by individual permissions. This simplifies the security definitions required.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000327

9.2 Basic security configuration

First we discuss our basic security configuration, taking a CICS region with no security and configuring it to enable transaction security (we do not implement other types of CICS security such as resource security and command security).

Note: The security scenarios in this chapter are based on a different MVS system that those documented in Chapter 8, "Security scenarios" on page 247. We therefore repeat some of the security setup steps for CICS region CIWSS3C1 here.

In this section, we document the following steps:

- Creating a RACF key ring
- System initialization table parameters necessary to set up basic CICS security
- Testing the basic security configuration
- Configuring the pipeline
- Setting a user ID on a URIMAP definition

9.2.1 Creating a RACF key ring

In order to use the CICS WS-Security support for signing and encryption, it is necessary to create a RACF key ring, public-private key pairs, and X.509 certificates. In 9.4, "Certificate and key pair generation" on page 319 we show how to create certificates and key pairs for the signing and encryption scenarios. In the basic CICS security setup, however, we created a key ring for use by CICS region CIWSS3C1.

Note: In our tests we found that the CICS region must be connected to a RACF key ring even if the only CICS WS-Security support being used is basic authentication.

The sample clist DFH$RING is provided by CICS TS to help you build a RACF key ring and sample X.509 certificates for SSL usage. We used the DFH$RING clist to create the key ring `Ciws.Ciwss3cl`. See 8.5.1, "Creating a key ring and certificates on z/OS for CICS" on page 274 for information on using the DFH$RING sample clist.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000328

9.2.2 Specifying the security SIT parameters

We specified the following security SIT parameters for our CICS region:

- KEYRING=CICS.CICSSDC

  This specifies the name of a key ring we created in the RACF database that contains keys and certificates used by this CICS region.

- SEC=YES

  SEC=YES was specified to indicate that we wanted RACF services to control access to CICS resources.

- SECPRFX=YES

  We used security prefixing (SECPRFX=YES) in our CICS region, which prevents our RACF security profiles from affecting other CICS regions. This is useful in a production environment since it means all security profiles are unique to an individual region; however, it can mean more work for the security administrator because more profiles must be defined.

- XTRAN=YES

  XTRAN=YES was specified so CICS would control who could start transactions.

- XUSER=YES

  XUSER=YES specifies that CICS is to perform surrogate user checking.

For detailed information on CICS security, see *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454.

9.2.3 Testing the basic security configuration

In our testing, we followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec2.ear.

When attempting to place an order with the catalog application, the Web browser received an error message and the cause of the error was reported by CICS as a security violation (Example 9-1).

Example 9-1 CICSUSER security violation messages during CPIH

```
DFHXS1111 08/09/2006 18:05:38 46P053CI CPIH Security violation by user CICSUSER for resource
          CICS3D.CPIH in class TCICSTRN. SAF codes are (X'0CCCCCC8',X'00000008'), ESM codes
          are (X'CCCCCCC8',X'00000000').
DFHWB0561 08/09/2006 18:05:38 46P053CI An attempt to attach a CICS Web alias transaction for
          userid CICSUSER has failed because the user is not authorized to execute transaction
          CPIH. Host IP address: 9.12.4.75. Client IP address: 9.12.4.201.
          TCPIPSERVICE: SSCI.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000329

```
DFHXS2003 09/09/2006 18:05:38 A6P053CL Security violation has been detected term id = ????,
          trans id = CPIH, userid = CICSUSER
```

The CICS default user ID does not have access to the CPIH transaction. This shows that the CICS default user ID is not authorized to access the placeOrder service. In the following scenarios, we enable placeOrder service access to a set of authorized user IDs.

9.2.4 Configuring the pipeline

Example 9-2 shows the inital pipeline configuration file used for our CICS WS-Security test setup

Example 9-2 Pipeline configuration file ITSO_7206_basicsoap12provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/hts/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/hts/cics/pipeline
  provider.xsd ">
    <transport>
      <default_transport_handler_list>
          <handler>
             <program>CIWSMSGH</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
    </transport>
    <service>
      <service_handler_list>
        <handler>
          <program>SHOFFER</program>
          <handler_parameter_list/>
        </handler>
      </service_handler_list>
      <terminal_handler>
        <cics_soap_1.2_handler/>
      </terminal_handler>
    </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

- The message handler program CIWSMSGH is used to change the transaction ID to INQS, INQC, or ORDR based on the service request. See "Customizing the pipeline configuration file" on page 80 for more information on the CIWSMSGH program.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000330

- The message handler program SNIFFER is a simple program that browses through the containers available in the pipeline. See "Using SNIFFER" on page 106 for more information on the SNIFFER program.

9.2.5 Setting a user ID on a URIMAP definition

We initially want to run the CPIH, INQS, INQC, and ORDR transactions under a pre-defined user ID, other than the default CICS user ID for the security scenarios that follow, we then show how to run ORDR transactions for specific placeOrder service requests under user IDs which flow in the SOAP messages

To set the pre-defined user ID for the CPIH and ORDR transactions, we created a URIMAP definition for the placeOrder service such that the transactions run with the user ID CICSNW. See Section 8.3, "Setting the user ID on a URIMAP definition" on page 252 for more information on defining URIMAPs. We then permitted RACF access to the CPIH and ORDR transactions for the user ID CICSNW.

Figure 9-1 shows the CPIH and ORDR transactions running under user ID CICSNW

```
INQUIRE TASK
STATUS:  RESULTS - OVERTYPE TO MODIFY
  Tas(0000082) Tra(CPIH)           Sta Tas Pri( 001 )
     Sta(U ) Use(CICSNW  ) Dsn(B=3BFA3A452A25D3C) Hty(DFHPO**)
  Tas(0000083) Tra(ORDR)           Sta Tas Pri( 001 )
     Sta(U ) Use(CICSNW  ) Dsn(B=3BFA3A474A2D9C9) Hty(EC*     )
  Tas(0000084) Tra(CEUT) Fac(EC22) Sta Ter Pri( 001 )
     Sta(SD) Use(WISCLM  ) Dsn(B=3BFA3A47A210343) Hty(ZCICWA**)
  Tas(0000085) Tra(CONT) Fac(CC21) Sta Ter Pri( 255 )
     Sta(TO) Use(WISCLM  ) Dsn(B=3BFA3A5A97D2631)
```

Figure 9-1 ORDR executing with preset user ID

We also created URIMAP resources for the inquireSingle and inquireCatalog services such that the INQS and INQC transactions also run with user ID CICSNW, and we permitted RACF access to the INQS and INQC transactions for the user ID CICSNW.

9.3 Basic authentication

WebSphere Application Server supports the generation of security tokens that are passed in the WS-Security SOAP header. A UsernameToken is a token for

Document ID: [illegible] Page 00450

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000331

basic authentication; it has a user name and password. In 8.4, "Enabling SOAP message security with HTTP" on page 257 we show how we wrote the message handler program CIWSSECS, which we then used to process a UsernameToken passed in a Web service call from WebSphere Application Server.

In this section we show how a UsernameToken can also be processed by the CICS-supplied message handler, DFHWSSE1. DFHWSSE1 extracts the UsernameToken from the SOAP header, validates the username and password, and sets the user ID of the CICS task to the username passed in the header.

Note: CICS supports UserNameToken in service provider mode only.

This scenario is shown in Figure 9-2.

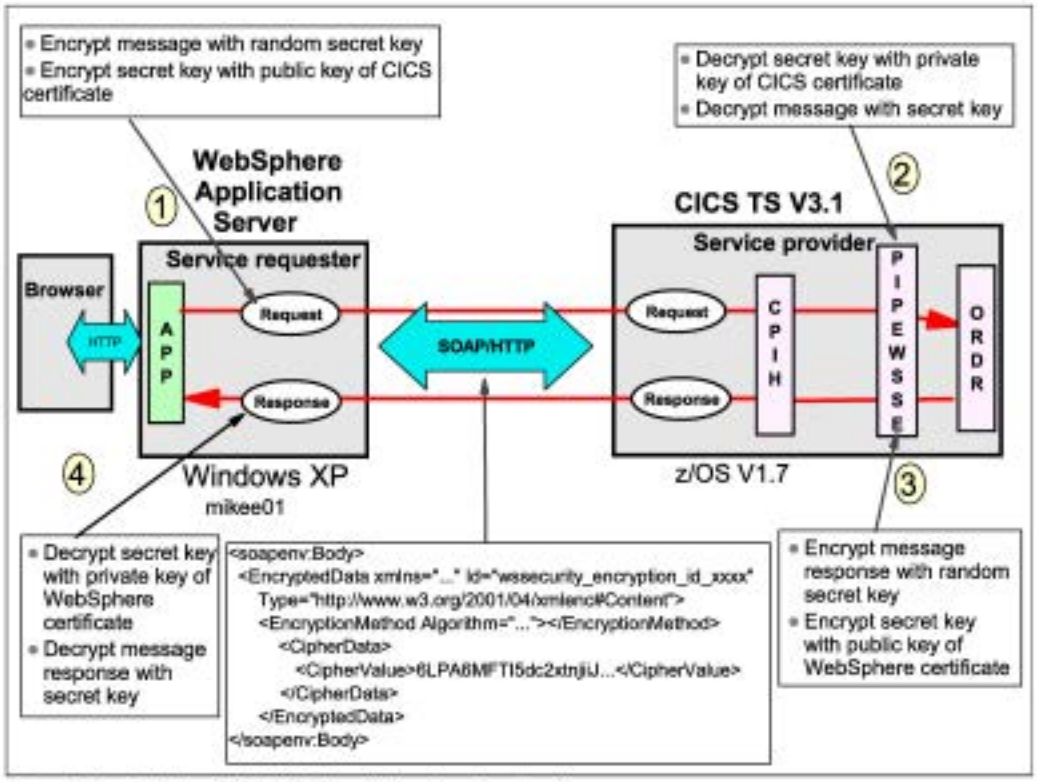


Figure 9-2 Enabling CICS WS-Security basic authentication

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 332/f9

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000332

Recommendation: Internal tests have shown that the performance of a user-written message handler program such as CIWSSECS for processing a UsernameToken is significantly better than DFHWSSE1. If throughput is expected to be high and performance is a priority, we recommend that you use a user-written message handler program for basic authentication. If, however, you do not want to write custom handlers, or if performance is not a priority, then enabling basic authentication using the CICS-supplied DFHWSSE1 is a simpler solution.

We document the following steps that we performed to enable support for basic authentication:

- Configuring the service requester for basic authentication
- Configuring CICS for basic authentication
- Testing the basic authentication scenario

9.3.1 Configuring the service requester for basic authentication

The service requester setup tasks that we cover here are as follows:

- Defining the WebSphere WS-Security constraint for the Web service client
- Re-deploying the Web service client application

WebSphere WS-Security configuration files

The WebSphere WS-Security constraints are defined in the IBM extension of the J2EE Web services deployment descriptor. There are four configuration files:

- Client deployment descriptor extension file - includes request generator and response consumer constraints:

  ibm-webservicesclient-ext.xmi

- Client binding configuration file - includes how to apply request generator and response consumer constraints:

  ibm-webservicesclient-bnd.xmi

- Server deployment descriptor extension file - includes request consumer and response generator constraints:

  ibm-webservices-ext.xmi

- Server binding configuration file - includes how to apply request consumer and response generator constraints:

  ibm-webservices-bnd.xmi

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000333

The deployment descriptor extension files specify *what* security constraints are required, for example, what type of security token and whether to sign the message. The binding files specify *how* to apply the required security constraints defined in the deployment descriptor extension, for example, which security token is inserted and which keys are used for signing.

These deployment descriptor extension and binding files define the application-level security constraints and they belong to each application. See "WebSphere and SOAP message security" on page 239 for more information on configuring WebSphere WS-Security.

For our scenario, we needed to configure the client deployment descriptor and the client binding only, since our service provider is running in CICS and not in WebSphere Application Server.

Importing the base application

To configure our Web Service client to use WS-Security, we used the IBM WebSphere Application Server Toolkit V6.1 (AST).

Note: You can also use Rational Application Developer V6.0 (RAD) to configure WS-Security for WebSphere applications.

We performed the following steps:

1. We opened the AST in a new workspace. In Windows we clicked:

   Start → Programs → IBM WebSphere → Application Server Toolkit V6.1 → Application Server Toolkit

   We specified a new workspace for this scenario called

   `C:\AST\CatalogSec2\CatalogSec2BasicAuth`

2. To import the CatalogSec2 base application we clicked the **File** menu and then **Import**.
3. We selected **EAR file** in the list of options and clicked **Next**.
4. We clicked **Browse** to select the CatalogSec2.ear file. This file contains the Catalog base application.
5. We also changed the EAR project name to CatalogSec2BasicAuth as shown in Figure 9-3.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000334



Figure 9-3 Importing the CatalogSec2 ear file

6. Finally, we clicked **Next** → **Next** → **Finish**.

Editing the client configuration

We performed the following steps to edit the client configuration:

1. We expanded the Dynamic Web Project (CatalogSec2Web) in the Project Explorer and opened (double-clicked) the deployment descriptor as shown in Figure 9-4.

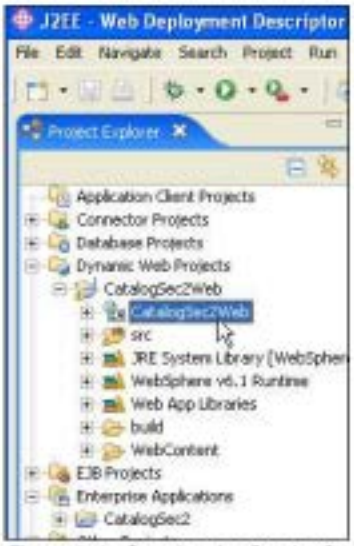


Figure 9-4 Opening the CatalogSec2Web Deployment Descriptor.

2. After the Deployment Descriptor Editor opened, we selected the **WS Extension** page. The WS Extension page is used for editing the client's

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 335/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000335

deployment descriptor extension file (ibm-webservicesclient-ext.xmi), so you can specify what security is required. Figure 9-5 shows the WS Extension page in the Deployment Descriptor Editor.

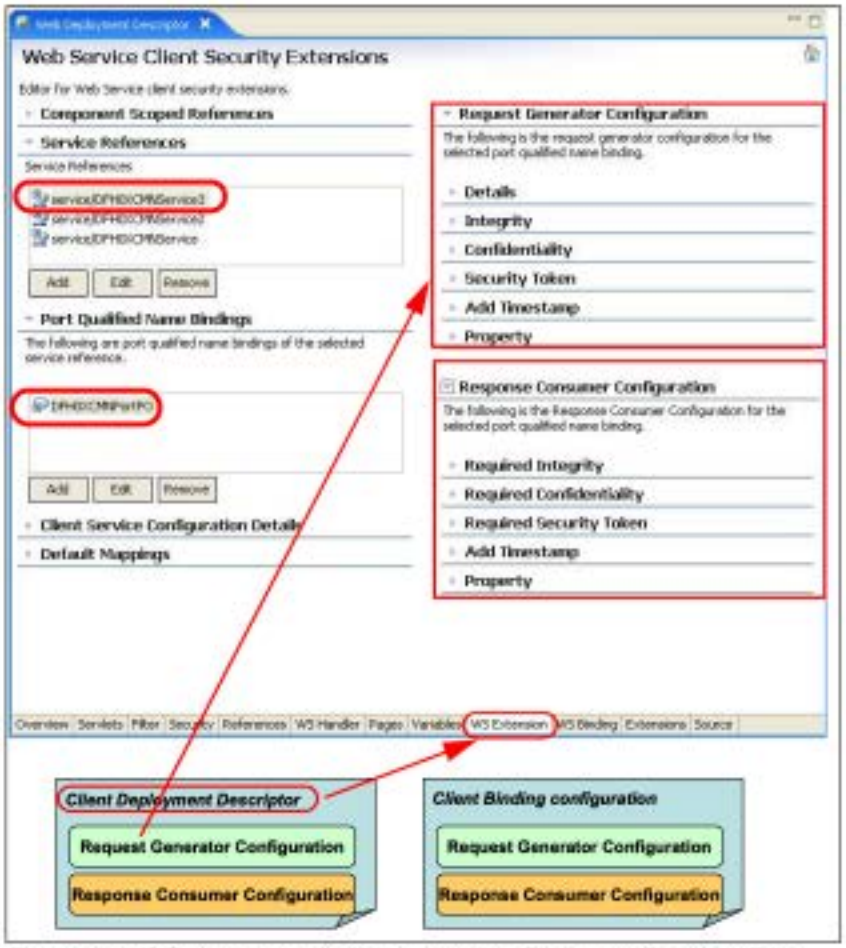


Figure 9-5 WS Extension page in Web service client Deployment Descriptor Editor

3. We clicked the **service/DFH0XCMNService3** reference (for the placeOrder service) and **DFH0XCMNPortPO** (the port binding for the service reference).
4. In the Request Generator Configuration, we expanded **Security Token** and clicked **Add**. We entered the name `basicauth`.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 336/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000336

5. We selected the **Username Token** token type from the drop-down list. When you select a Token type, the Local name is filled in automatically (Figure 9-6). We left the URI empty.



Figure 9-6 Security Token Dialog for specifying basic authentication

6. We clicked **OK** and a security token was created. We then saved the configuration by pressing Ctrl+s.
7. After specifying the security token, a corresponding token generator must be specified in the binding configuration. The WS Binding page is for editing the client's binding file, so you can specify how to apply the required security. We clicked the **WS Binding** tab. Figure 9-7 shows the WS Binding page in the Deployment Descriptor Editor.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 337/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000337

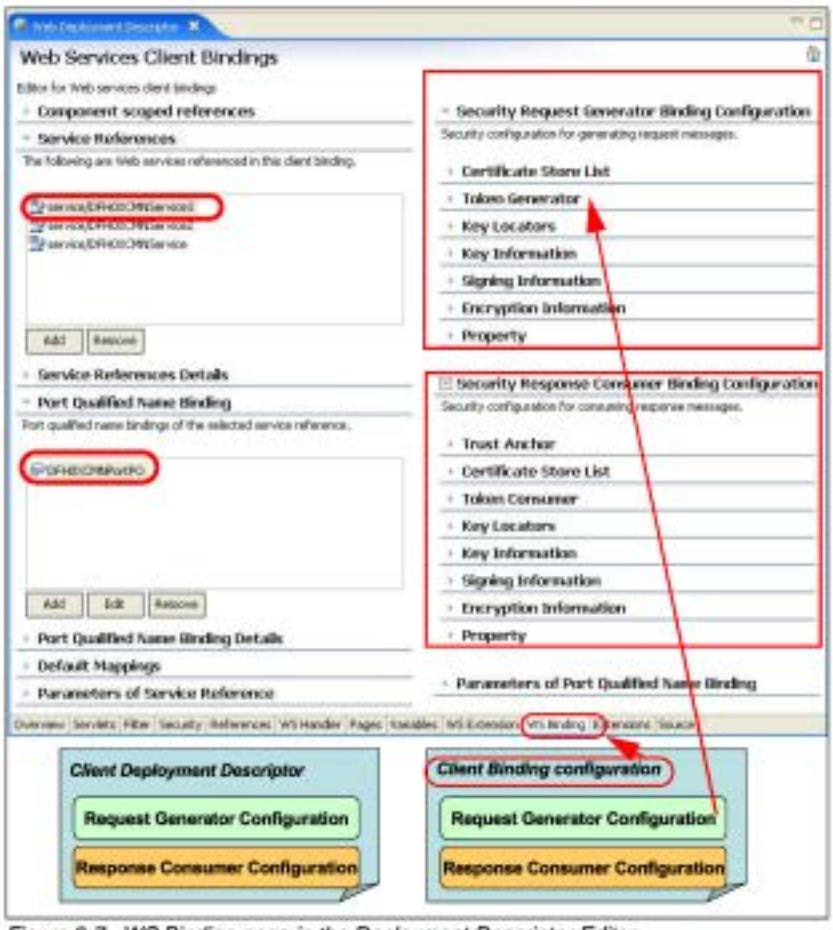


Figure 9-7 WS Binding page in the Deployment Descriptor Editor

8. We selected **Token Generator** and clicked **Add**. We entered a Token generator name `basicauthToken` and allowed the Token generator class to default as `com.ibm.wsspi.wssecurity.token.UsernameTokenGenerator`
9. To select the security token, we expanded the drop-down list and selected the **basicauth** security token that we defined previously on the WS Extension page.
10. We checked **Use value type** and selected the **Username Token** value type from the drop-down list. This selection filled the local name and callback handler.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 338/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000338

11. We entered the user ID WEBAS1 and password REDBOOKS that we wanted to use to identify the Web service client (Figure 9-8).

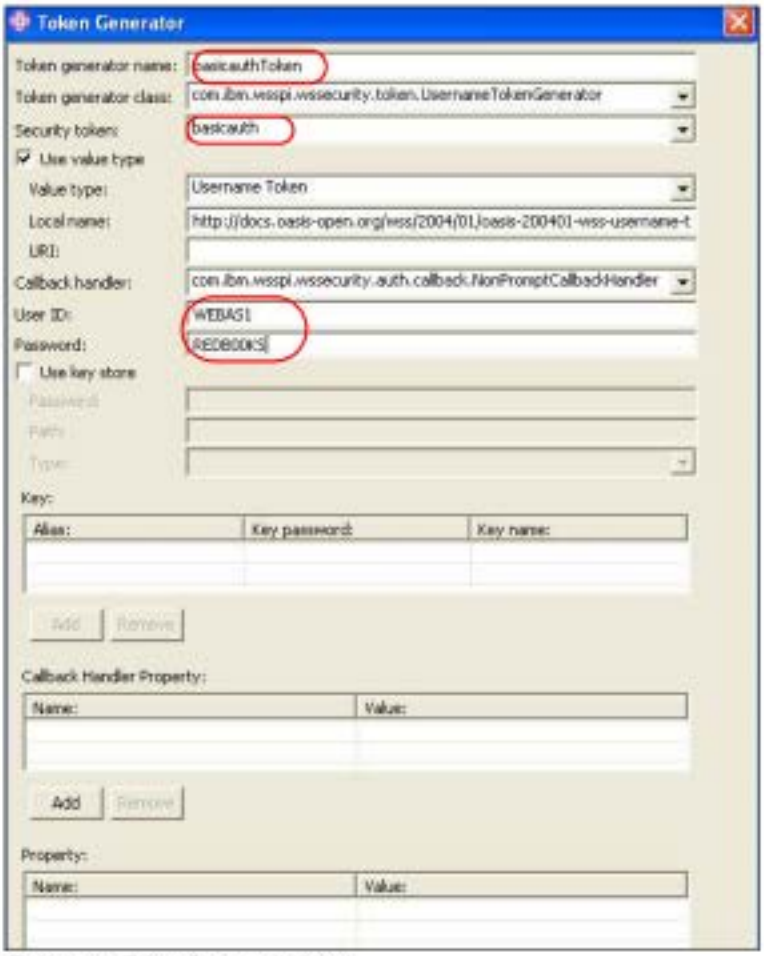


Figure 9-8 Token Generator dialog

12. We clicked **OK**, and the token generator was created. We then saved the configuration.

Note: The User ID and Password in Figure 9-8 represent the credentials of the application server, *not* the browser user. The same credentials are passed to CICS in the SOAP message irrespective of the identity of the end user.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 339/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000339

Redeploying the Web service application

After configuring a basic authentication security constraint for the service requester application, we exported a new EAR file called CatalogSec2_WS-Security_BasicAuth.ear. We followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec2_WS-Security_BasicAuth.ear.

For further information on configuring WS-Security in WebSphere, refer to *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257.

9.3.2 Configuring CICS

The credentials configured using the Application Server Toolkit are used to form the WS-Security header that is contained in the SOAP message. An example of this message is shown in Example 9-3.

Example 9-3 SOAP security header extracted from a SOAP message

```
<wsse:Security soapenv:mustUnderstand="1"
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd">
    <wsse:UsernameToken>
      <wsse:Username>WEBAS1</wsse:Username>
      <wsse:Password Type="http://docs.oasis-open.org/wss/2004/01/oasis-200401
       -wss-username-token-profile-1.0#PasswordText">REDBOOKS</wsse:Password>
    </wsse:UsernameToken>
</wsse:Security>
```

This header includes the mustUnderstand="1" attribute, which indicates that either this header must be processed or a SOAP fault thrown.

In order for CICS to process the security token in the SOAP header of the placeOrder service request, we made the following changes to the CICS configuration:

- Created a pipeline configuration file for basic authentication
- Created a new pipeline to process placeOrder requests and altered the URIMAP for the placeOrder service to use the new pipeline
- Permitted access to the ORDR transaction for user ID WEBAS1
- Permitted surrogate access for the user ID WEBAS1

The details for those procedures are presented in the following sections

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000340

Creating a pipeline configuration file for basic authentication

Example 9-4 shows the basic authentication pipeline configuration file that includes the message handler DFHWSSE1, and the configuration information for the handler.

Example 9-4 Pipeline config file, ITSO_7206_wsse_basicauth_provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/txseries/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/txseries/cics/pipeline
  provider.xsd ">
    <transport>
      <default_transport_handler_list>
          <handler>
             <program>CWSMSG</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
    </transport>
    <service>
      <service_handler_list>
        <handler>
          <program>SMI0000</program>
          <handler_parameter_list/>
        </handler>
        <wsse_handler>
          <dfhwsse_configuration version="1">
            <authentication mode="basic">
            </authentication>
          </dfhwsse_configuration>
        </wsse_handler>
      </service_handler_list>
      <terminal_handler>
        <cics_soap_1.2_handler/>
      </terminal_handler>
    </service>
  <appHandler>DFHPITP</appHandler>
</provider_pipeline>
```

The <wsse_handler> element shown in Example 9-4 contains a <dfhwsse_configuration> element that specifies configuration information for DFHWSSE1.

The <authentication mode="basic"/> specifies that inbound messages must contain a UserNameToken.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000341

Creating a new pipeline to process placeOrder requests

Since the inquireCatalog and inquireSingle service requests will not be subject to basic authentication, we need to create a new pipeline specifically for the placeOrder requests. We also need to change the SECPORDR URIMAP definition to link the placeOrder service to the new pipeline. We do this using the RDO commands in Example 9-5 and then re-installing the PIPELINE and URIMAP resource definitions.

Example 9-5 Create pipeline for WS-Security basic authentication

```
CEDA COPY PIPELINE(PIPE1) GROUP(SSCTEXMS) AS(PIPEWSSE)
CEDA ALTER PIPELINE(PIPEWSSE) GROUP(SSCTEXMS)
configfile(/CICS/SSCT/configs/CICS_FZIB_wssec_basicauth_provider.xml)
CEDA ALTER URIMAP(SECPORDR) GROUP(SSCTEXMS) PIPELINE(PIPEWSSE)
```

Permitting access to the ORDR transaction

We permitted access to the ORDR transaction for the user ID WEBAS1 with the following:

```
PERMIT CICS3D.ORDR CLASS(TCICSTRN) ID(WEBAS1) ACCESS(READ)
SETROPTS RACLIST(TCICSTRN) REFRESH
```

Permitting surrogate access

Example 9-6 shows the RACF commands that we used to permit the pre-defined user ID (CICSNW) to start transactions (such as ORDR) with the user ID WEBAS1.

Example 9-6 RACF commands to allow CICSNW to act as surrogate for WEBAS1

```
RDEFINE SURROGAT WEBAS1.DFHSTART UACC(NONE) OWNER(MAINT)
PERMIT WEBAS1.DFHSTART CLASS(SURROGAT) ID(CICSNW) ACCESS(READ)
SETROPTS RACLIST(SURROGAT) REFRESH
```

For more information regarding surrogate user security, see *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454.

9.3.3 Testing basic authentication

Now when we submit an order request on the Web browser, the credentials are added to the SOAP Security header and are extracted by the DFHWSSE1 program.

Figure 9-9 shows the ORDR transaction running with the user ID WEBAS1, while the CPIH transaction continues to execute with the CICSNW user ID.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000342

```
INQUIRE TASK
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tas(0000055) Tra(CPIH)           Sus Tas Pri( 001 )
    Sta(U ) Use(CIWSMW  ) Uow(BF3C2E9E2114284E) Hty(RZCBNOTI)
 Tas(0000056) Tra(ORDR)           Sus Tas Pri( 001 )
    Sta(U ) Use(WEBAS1  ) Uow(BF3C2E9E24801B8B) Hty(EDF     )
 Tas(0000058) Tra(CEDF) Fac(E024) Sus Ter Pri( 001 )
    Sta(SD) Use(NIGELW  ) Uow(BF3C2E9E28D4A149) Hty(ZCIOWAIT)
 Tas(0000059) Tra(CEMT) Fac(E025) Run Ter Pri( 255 )
    Sta(TO) Use(NIGELW  ) Uow(BF3C2EA64198CCCE)

                                                SYSID=S3C1 APPLID=A6POS3C1
```

Figure 9-9 ORDR executing with basic authentication user ID

SOAP fault messages

In this section we show some of the SOAP faults that you may see after enabling CICS WS-Security basic authentication.

- Service requester does not send UserNameToken

  If the service requester does not send a UsernameToken, DFHWSSE1 returns a SOAP fault as shown in Figure 9-10.

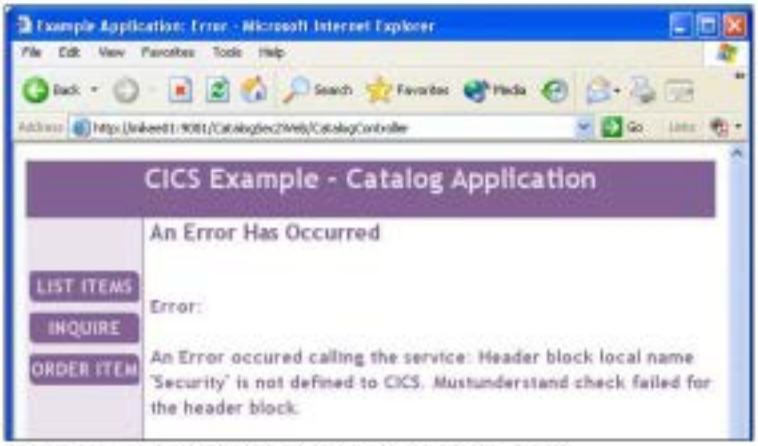


Figure 9-10 DFHWSSE1 SOAP fault: No UsernameToken

- UserNameToken credentials are invalid

  If the credentials extracted do not validate, then DFHWSSE1 creates a SOAP fault message as shown in Figure 9-11.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 343/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000343

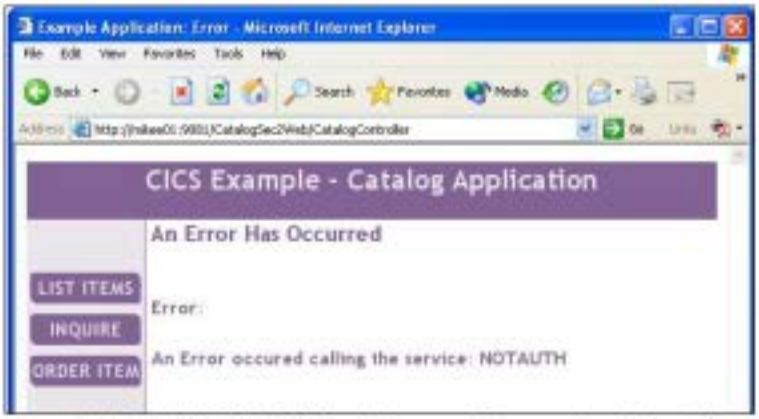


Figure 9-11 DFHWSSE1 SOAP fault: UsernameToken credentials invalid

The SOAP fault for this message, shown in Example 9-7, highlights that the user credentials are invalid and that the RACF revoke count is incremented for this user ID.

Example 9-7 SOAP fault for invalid basic authentication credentials

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>wsse:FailedAuthentication
      </faultcode>
      <faultstring>NOTAUTH
      </faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
          <message>The supplied password is wrong. If the external security
          manager is RACF, the revoke count maintained by RACF is incremented
          </message>
          <errorcode>3
          </errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 344/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000344

- WebSphere rejects SOAP response sent by CICS due to empty security header.

  During our testing we experienced the problem shown in Figure 9-12.

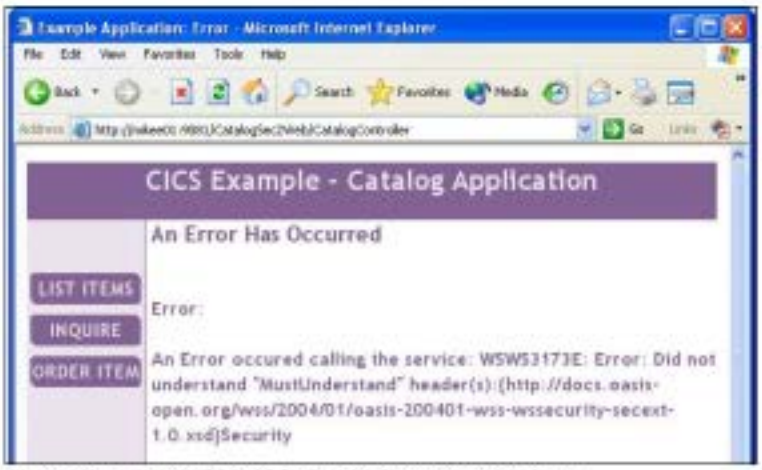


Figure 9-12 DFHWSSE1 SOAP fault: Empty security token

In this case CICS has returned the empty security header shown in Example 9-8.

Example 9-8 Empty security header returned from CICS

```
<wsse:Security
xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" SOAP-ENV:mustUnderstand="1"
xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-u
tility-1.0.xsd" xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"/>
```

We found that, depending on the version of WebSphere Application Server and the maintenance level of CICS TS V3.1, it is possible that WebSphere will reject this SOAP response from CICS due to it containing an empty security header.

The WebSphere exception for this error, shown in Example 9-9, highlights that WebSphere is not expecting a security header and is rejecting the empty one sent by CICS because it contains the `mustUnderstand="1"` attribute.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 345/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000345

Example 9-9 WebSphere exception for empty security header

```
Exception: WSWS3173E: Error: Did not understand "MustUnderstand"
header(s):{http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-s
ecext-1.0.xsd}Security:
```

We were able to bypass this problem by configuring WebSphere to optionally receive a security token sent in the CICS response.

Perform the following steps to configure WebSphere to tolerate an empty security header sent in the CICS response:

1. Open the CatalogSec2Web Deployment Descriptor.
2. Select the WS Extension page.
3. In Service References select the **service/DFH0XCMNService3** reference. then click **DFH0XCMNPortPO** in Port Qualified Name Bindings.
4. Expand **Response Consumer Configuration**.
5. Expand **Required Security Token** and click **Add**.
6. In the Required Security Token window returned (Figure 9-13), make the following entries and selections:
   - Enter a name for this required security token, for example, `requntoken`.
   - Select **Username Token** from the Token type drop-down list.
   - Select **Optional** for the Usage type from the drop-down list.
     The available choices are Required or Optional. If you select Required, a SOAP fault is thrown if a required security token is not included in a client's request message. If you select Optional, the process of consuming a request message continues without throwing a SOAP fault.
7. Click **OK**.



Figure 9-13 WS Extension page Required Security Token dialog box

8. Save the configuration by pressing Ctrl+s.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 346/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000346

9. Open the WS Binding page.
10. In Service References select the **service/DFH0XCMNService3** reference, then click **DFH0XCMNPortPO** in Port Qualified Name Bindings.
11. Expand **Security Response Consumer Binding Configuration.**
12. Expand **Token Consumer** and click **Add.**
13. In the Token Consumer dialog box returned (Figure 9-14) make the following entries and selections:
    - Enter a Token consumer name, for example `con_untcon`.
    - Select **com.ibm.wsspi.wssecurity.token.UsernameTokenConsumer** as the Token consumer class.
    - From the Security token drop-down list select the security token specified in the WS Extension page. In our case this was **requntoken**.
    - Select the **Use jaas.config** option and enter `system.wssecurity.UsernameToken` as the jaas.config name.
14. Click **OK**.

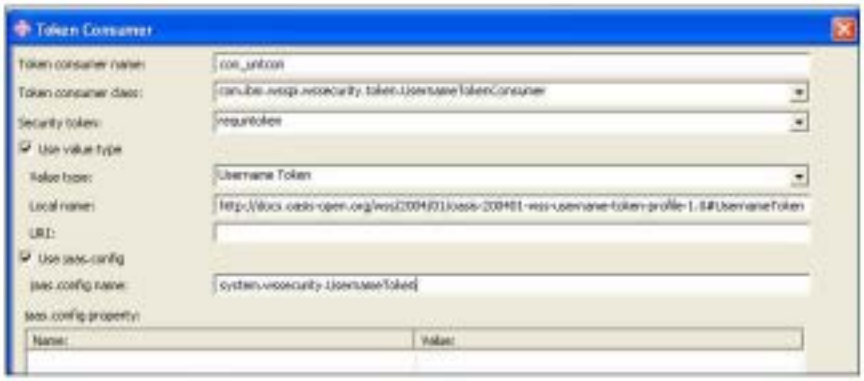


Figure 9-14 Token Consumer dialog box in WS Binding page

15. Save the Deployment Descriptor.
16. Export the application as an ear file and install it in the WebSphere Application Server.

Important: APAR PK31924 has been raised for this problem. If you experience this problem and no fix is available for PK31924 at the time, you will need to configure WebSphere to tolerate an empty security header sent by CICS. This problem can also occur for digital signature processing.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 347/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000347

9.4 Certificate and key pair generation

In this section we show how we created the certificates and key pairs that are used in the WS-Security signature and encryption test scenarios that follow.

When setting up a WS-Security environment you have a choice between self-signed certificates and CA-signed certificates:

- Self-signed certificates

  A self-signed certificate is an identity certificate that is signed by its creator, so the creator is verifying that the certificate is valid.

- Certificating authority (CA) signed certificates

  CA-signed certificates are created by a user organization and sent to a CA to be signed. Before signing a certificate the CA verifies that the organization requesting the certificate is actually who they claim to be, so the CA is verifying that the certificate is valid.

In the examples in this chapter, we use self-signed certificates because they are more easily and quickly generated. In a production environment, it is expected that CA-signed certificates will be used because they provide a more secure solution.

Figure 9-15 shows the certificates and key pairs used in the signature and encryption test scenarios.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000348

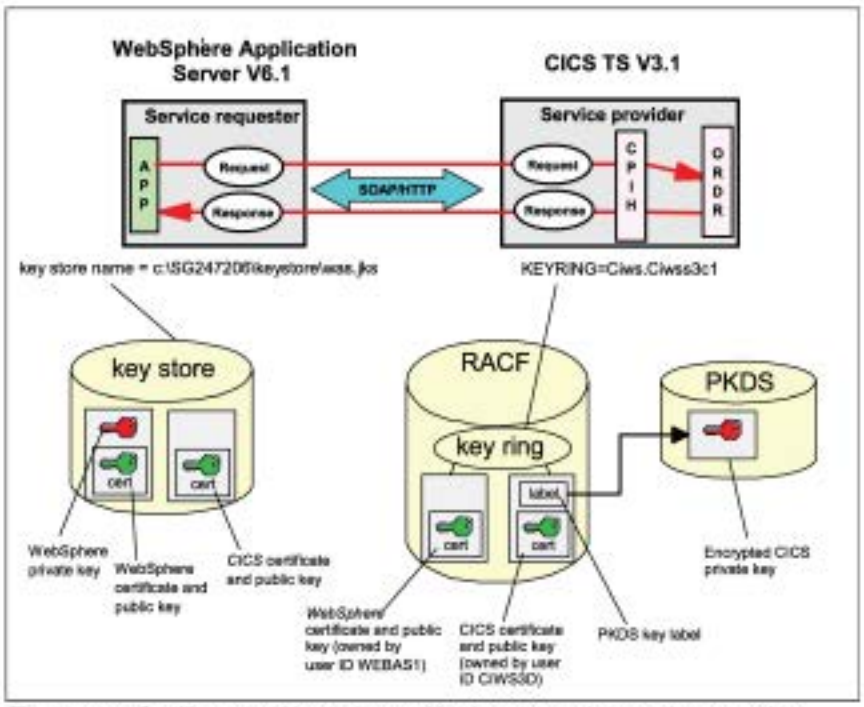


Figure 9-15 Certificates and key pairs used in signature and encryption scenarios

Figure 9-15 shows two stores for our certificates and key pairs:

- The WebSphere key store
  - The WebSphere key store contains the WebSphere certificate and holds the WebSphere public and private key pair.
  - The key store also contains the CICS certificate with the public key only.
- RACF
  - RACF contains the CICS certificate and holds the CICS public key. The associated private key is stored in a Private Key Data Set (PKDS).

    In a public key cryptographic system, it is a priority to maintain the security of the private key. It is vital that only the intended user or application have access to the private key.

    ICSF and the cryptographic hardware features ensure this by enciphering PKA private keys under a unique PKA object protection key. The PKA object protection key has itself been enciphered under a PKA *master key*. Each PKA private key also has a name that is cryptographically bound to

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 349/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000349

the private key and cannot be altered. ICSF uses the private key name or the *PKDS key label* to control access to the private key. This combination of hardware-enforced coupling of cryptographic protection and access control, through the use of RACF, is unique to ICSF. It provides a significant level of security and integrity for PKA applications.

– RACF also contains the WebSphere certificate with the public key only.

Note: If you use CA-signed certificates, you also need to add the CA certificates to the WebSphere key store and RACF key ring.

In this section, we document the following steps:

- Validating the cryptography hardware environment
- Creating the CICS certificate and key pair
- Creating the key store, WebSphere certificate, and key pair
- Adding the CICS certificate to the key store
- Adding the WebSphere certificate to RACF

Validating the cryptography hardware environment

CICS support for WS-Security signature and encryption processing benefits from the hardware acceleration capabilities of System z. It is a requirement, therefore, to run CICS on a system that has the appropriate cryptographic hardware configured. The specific cryptographic hardware requirement is dependent on the server type and z/OS level. See 6.12.1, "Cryptographic hardware requirements for CICS WS-Security" on page 211 for more information on cryptographic hardware requirements.

The SC66 host system used for our signature and encryption test scenarios was a z9 EC (Enterprise Class) with 18 CPs. The SC66 had the following cryptographic hardware functions enabled:

- CP Assist for Cryptographic Function (CPACF)
- Crypto Express2 (CEX2) feature

Three CEX2C coprocessors and one CEX2A accelerator were configured.

Important: The CEX2 feature requires ICSF to be active.

Integrated Cryptographic Service facility

The Integrated Cryptographic Service Facility (ICSF) is a software element of z/OS that works with hardware cryptographic features and RACF to provide

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 350/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000350

secure, high-speed cryptographic services in the z/OS environment. See 6.12, "Integrated Cryptographic Service Facility" on page 210 for more information.

Example 9-10 shows the startup messages of ICSF on the SC66 system. It shows that 3 CEX2C coprocessors and 1 CEX2A coprocessor are configured.

Example 9-10 ICSF startup messages

```
S CSF
IEF695I START CSF WITH JOBNAME CSF IS ASSIGNED TO USER STC, GROUP SYST
IEE402I CSF    STARTED   ASID=0096   SC66
CSFM441I CRYPTO EXPRESS2 COPROCESSOR E01, SERIAL NUMBER 94000264, ACTIVE
CSFM441I CRYPTO  EXPRESS2 COPROCESSOR E02, SERIAL NUMBER 95001434, ACTIVE
CSFM441I CRYPTO EXPRESS2 COPROCESSOR E03, SERIAL NUMBER 95001437, ACTIVE
CSFM135I CRYPTO EXPRESS2 ACCELERATOR F00 IS ACTIVE
CSFM001I ICSF INITIALIZATION COMPLETE
CSFM400I CRYPTOGRAPHY - SERVICES ARE NOW AVAILABLE.
```

ICSF must be configured and started in order to create public/private key pairs to be used by CICS for XML signature or encryption processing.

Creating the CICS certificate and key pair

In order to enable signature and encryption processing, we need to create a certificate and key pair to be used by CICS, and then connect the certificate to the CICS key ring.

Example 9-11 shows the RACF command that we used to create the CICS X.509 certificate.

Example 9-11 Create the CICS X.509 certificate (CICSCERT)

```
RACDCERT ID(CICSSOJ) GENCERT
 SUBJECTSDN(CN('CICSCERT')
            T('CICSSOJ-cert')
            OU('PSSC')
            O('ITSO')
            L('Endicott')
            SP('New York')
            C('US'))
 NOTBEFORE(DATE(2005-01-01) TIME(00:00:00))
 NOTAFTER(DATE(2010-12-31) TIME(23:59:59))
 KEYUSAGE(DOCSIGN DATAENCRYPT)
 WITHLABEL('CICSCERT')
 SIZE(1024)
 TCSF
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000351

- The RACDCERT GENCERT command creates the certificate and a public/private key pair.
- ID specifies that the CICS region user ID CIWS3D is to be associated with the certificate.
- SUBJECTDSN specifies the subject's X.509 distinguished name.
- NOTBEFORE and NOTAFTER specify the certificate validity range.
- KEYUSAGE specifies how the keys associated with the certificate are to be used. We specify:
  - DOCSIGN specifies that the certificate will be used for signing.
  - DATAENCRYPT specifies that the certificate will be used for encryption.
- WITHLABEL specifies the label name to be associated with the certificate.
- SIZE specifies the size of the private key expressed in decimal bits. We specified a high strength key length of 1024.
- ICSF specifies that the private key to be generated is an RSA key to be stored in the ICSF PKDS (public key data set).

  A certificate that CICS uses to sign SOAP messages (or decrypt encrypted SOAP messages) must be created with the ICSF option and ICSF must be configured and started in order for the certificate to be created or used by CICS. If ICSF is not operational, or does not have access to any cryptographic coprocessors or accelerators, you will receive the message `ICSF is not operational` when attempting to create the X.509 certificate.

Important: A certificate that CICS uses to sign SOAP messages, or to decrypt encrypted SOAP messages, must be created with the ICSF option.

Example 9-12 shows the RACF commands that we used to connect the CICS X.509 certificate to the key ring and to list the certificate.

Example 9-12 RACF command to connect CICS certificate to RACF key ring

```
RACDCERT ID(CIWS3D) CONNECT(ID(CIWS3D) LABEL('CICSCERT') RING(Ciws.Ciwss3cl))
RACDCERT ID(CIWS3D) LIST
```

Example 9-13 shows the output of the RACDCERT LIST command.

Example 9-13 Listing of CICS X.509 certificate (CICSCERT)

```
Digital certificate information for user CIWS3D:

  Label: CICSCERT
  Certificate ID: 2QbDyebi88TDycP1w8XZ40BA
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Page 352/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000352

```
Status: TRUST
Start Date: 2005/01/01 00:00:00
End Date:   2014/12/31 23:59:59
Serial Number:
     >00<
Issuer's Name:
     >CN=CICSCERT.1 Created cert.OU=PSSC.O=IISO... ENDCERT.SP=NEW YORK,C=US
     <
Subject's Name:
     >CN=CICSCERT.1 Created cert.OU=PSSC.O=IISO...ENDCERT.SP=NEW YORK,C=US
     <
Key Usage: DATAENCRYPT, DOCSIGN
Private Key Type: ICSF
Private Key Size: 1024
PKDS Label: IRR.DIGTCERT.CCU530.SC00.BF3B6F434ED31FC9
Ring Associations:
  Ring Owner: CICSSC
  Ring:
     >Cics.CreateJet<
```

Creating the key store and WebSphere certificate

In order to enable signature and encryption processing, we need to create a certificate and key pair to be used by WebSphere. The certificate and key pair are stored in a key store.

Table 9-4 shows the data used to create the key store and the WebSphere certificate.

Table 9-4 Client key store and certificate values

Field	Value
Key store name	wasks
Filename	was.jks
Storepass	iiso
Storetype	JKS
Distinguished Name	CN=wascert, O=IBM, C=US
Key size	1024
Alias	wascert_rsa
Certificate file	wascert_rsa.cer

Both the WebSphere public and private keys are associated to the certificate.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000353

We performed the following tasks using the WebSphere Administration Console:

- Generating the client key store
- Creating the client certificate
- Extracting the client certificate file

Note: The capability to create certificates and key stores using the WebSphere admin console is new in WebSphere Application Server V6.1. If you are using an earlier version of WebSphere, you can use a key management tool such as *iKeyMan* or *keytool*. See "Importing the certificate" on page 277 for an example of using iKeyMan.

Creating the client keystore

Perform the following steps to create the client key store:

1. Select **Security → SSL certificate and key management**.
2. Under Related Items select **Key stores and certificates**.
3. The resulting panel shows you all the keystores that you have configured through the administrative console. Click **New** and enter these values:

Name:	`wasks`
Path:	`c:\SG247206\keystore\was.jks`
Password:	`itso`
Type:	`JKS`

4. Click **OK**.

Creating the client certificate

Use the following steps to create the personal certificate for the client key store:

- Select the **wasks** key store.
- Under Additional Properties click **Personal certificates**.
- Click **Create self-signed certificate** and enter these values:

Alias:	`wascert_rsa`
Key size:	`1024`
Common name:	`wascert`
Validity Period:	`365 (days)`
Organization:	`IBM`
Country:	`US`

- Click **OK**.

Be sure to save the configuration changes in the administrative console.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000354

The Java **keytool** command can be used to list the contents of a key store:

```
keytool -list -keystore was.jks -storepass itso -v
```

The keytool command comes with the Java runtime environment. You might need to add the path to your java jre using the command.

```
set PATH=<WAS_HOME>\java\jre\bin;%PATH%
```

In this command, <WAS_HOME> is the directory where your WebSphere is installed (usually C:\Program Files\IBM\WebSphere\AppServer).

Adding the CICS certificate to the WebSphere key store

To make the public key of the CICS certificate available to WebSphere, we performed the following steps:

1. We used the RACF command shown in Example 9-14 to extract the certificate from the RACF database to a sequential data set.

Example 9-14 RACF command to extract CICS certificate from RACF

```
RACDCERT ID(CICSUSR1) EXPORT (LABEL('CICSCERT')) DSN('CICS.CICSCERT.DER')
FORMAT(CERTDER)
```

The RACDCERT EXPORT command exports the certificate with label 'CICSCERT' to the MVS data set CICS.CICSCERT.DER as a DER encoded X.509 certificate.

2. We FTPed the CICS certificate to the WebSphere machine.

Example 9-15 shows how we used FTP to send the exported certificate (in binary format) to our WebSphere machine.

Example 9-15 FTP CICS certificate to WebSphere machine

```
C:\Documents and Settings\Administrator>ftp wtsc66.itso.ibm.com
Connected to wtsc66.itso.ibm.com.
220-FTPMVS1 IBM FTP CS V1R7 at wtsc66.itso.ibm.com, 21:50:57 on 2006-08-15
220 Connection will close if idle for more than 5 minutes.
User (wtsc66.itso.ibm.com:(none)): MIGFLM
331 Send password please.
Password:
230 MIGFLM is logged on.  Working directory is "MIGFLM.".
ftp> lcd c:\SG24\205\keystore
Local directory now C:\SG24\205\keystore.
ftp> bin
200 Representation type is Image
ftp> cd 'CICS'
250 "CICS." is the working directory name prefix.
ftp> get CICSCERT.CER
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000355

```
200 Port request OK.
125 Sending data set CINS.CICSCERT.DER
250 Transfer completed successfully.
ftp: 740 bytes received in 0.00Seconds 740000.00Kbytes/sec.
ftp> bye
221 Quit command received.
Goodbye.
```

3. We imported the CICS certificate into the WebSphere key store using the admin console.

   Figure 9-16 shows the certificate being imported using the admin console.

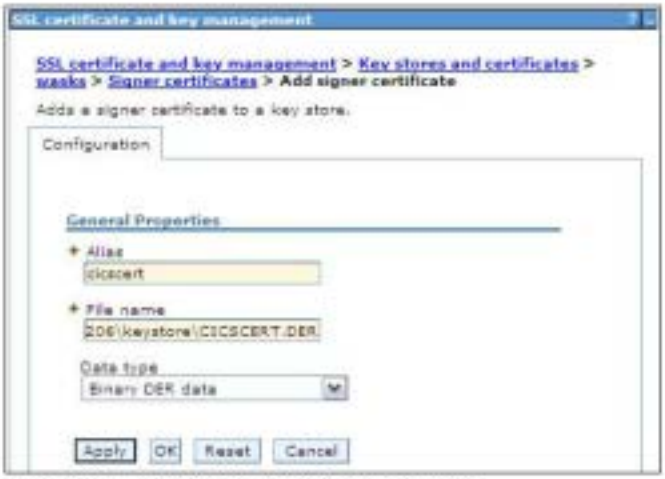


Figure 9-16 Import CICS certificate into key store

Adding the WebSphere certificate to RACF

Perform the following steps to make the public key of the WebSphere certificate available to CICS:

1. Extract the certificate from the key store using the WebSphere administration console.

   In "Creating the key store and WebSphere certificate" on page 324 we show how we created the WebSphere certificate. We now extract the certificate from the jks to file wascert.der (a Binary DER file).

   The certificate has both public and private keys. However, if you distribute the certificate, you must ensure that you only send the public key and not the private key.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 356/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000356

Follow these steps to export the public key from the wasks key store:

a. Locate the client personal certificates by selecting **SSL certificate and key management → Key stores and certificates → wasks → Personal certificates**.

b. Select the check box next to the newly created certificate `wascert_rsa`.

c. Click **Extract**.

d. For Certificate file name, specify `c:\SG247206\keystore\wascert.der`

e. For Data Type select **Binary DER data**.

f. Click **OK**.

Figure 9-17 shows the certificate being extracted using the admin console.



Figure 9-17 Extract client X.509 certificate from key store

2. FTP the WebSphere certificate to the host system.

   Before performing the FTP, we first allocated a sequential data set on the host machine SCTS66 to receive the certificate. This data set is allocated with a record format VB, record length 84 and block size 27998.

   We then used FTP as shown in Example 9-16 to send the exported certificate (in binary format) to our host machine.

Example 9-16 FTP client X.509 certificate to host

```
C:\Documents and Settings\Resident>ftp wtsc66.itso.ibm.com
Connected to wtsc66.itso.ibm.com.
220-FTPMVS1 IBM FTP CS V1R7 at wtsc66.itso.ibm.com, 21:04:02 on 2006-08-11.
220 Connection will close if idle for more than 5 minutes.
User (wtsc66.itso.ibm.com:(none)): NIGELW
331 Send password please.
Password:
```


Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 357/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000357

```
230 MIGELW is logged on.  Working directory is "MIGELW.".
ftp> lcd C:\SG247206\keystore
Local directory now C:\SG247206\keystore.
ftp> cd 'CICS'
250 "CICS." is the working directory name prefix.
ftp> bin
200 Representation type is Image
ftp> put wascert.der
200 Port request OK.
125 Storing data set CICS.WASCERT.DER
250 Transfer completed successfully.
ftp: 580 bytes sent in 0.00Seconds 580000.00Kbytes/sec.
ftp> bye
221 Quit command received.
Goodbye.
```

3. Import the client certificate into RACF and connect it to the CICS key ring.

   We used the RACF commands shown in Example 9-17.

Example 9-17 RACF commands to import WebSphere certificate into RACF

```
RACDCERT ID(WEBAS1) ADD('CICS.WASCERT.DER') WITHLABEL('WASCERT') TRUST
RACDCERT ID(CICSUSR) CONNECT(ID(WEBAS1) LABEL('WASCert') RING(Cics.Cicsdc1))
```

   - The RACDCERT ADD command adds the client certificate with label 'WASCERT' as a trusted certificate.
   - The RACDCERT CONNECT command connects the client certificate to the CICS key ring. ID(WEBAS1) indicates that the certificate being added to the key ring is a user certificate, and WEBAS1 is the user ID that is associated with this certificate.

   The certificate can be listed using this command:

```
RACDCERT ID(WEBAS1) LIST
```

   Example 9-18 shows the output of the RACDCERT LIST command.

Example 9-18 Listing of client X.509 certificate (WASCERT)

```
Digital certificate information for user WEBAS1:

  Label: WASCERT
  Certificate ID: 2QbuuC.B4xHxwLDado]
  Status: TRUST
  Start Date: 2006/08/01 14:36:07
  End Date:   2007/08/01 14:36:07
  Serial Number:
       >4A3A3C<
  Issuer's Name:
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000358

```
      "CN=wascert,OU=,O=IBM,L=,SP=,,C=US"
Subject's Name:
      "CN=wascert,OU=,O=IBM,L=,SP=,,C=US"
Private Key Type: None
Ring Associations:
  Ring Owner: CICSSD
  Ring:
     "Ciws.Ciwss3cl"
```

9.5 Signing a SOAP message

To provide integrity on the client's request message, we can add an XML digital signature to the message. CICS supports digital signatures in both service provider and service requester modes.

In this section we show how a SOAP request message can be signed by WebSphere Application Server and how the signature can be verified by CICS and how the response message can be signed by CICS and verified by WebSphere.

- The SOAP request message contains an X.509 certificate (the WebSphere certificate) that is used to sign the body of the message.
- The SOAP response message also contains an X.509 certificate (the CICS certificate) that is used to sign the body of the message.

The X.509 certificates are transported within a *BinarySecurityToken* element.

We also show how the WebSphere certificate can be used to authenticate the service requester. DFHWSSE1 verifies the signature and uses the certificate to determine the user ID under which the ORDR transaction will be run. This scenario is shown in Figure 9-18.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000359

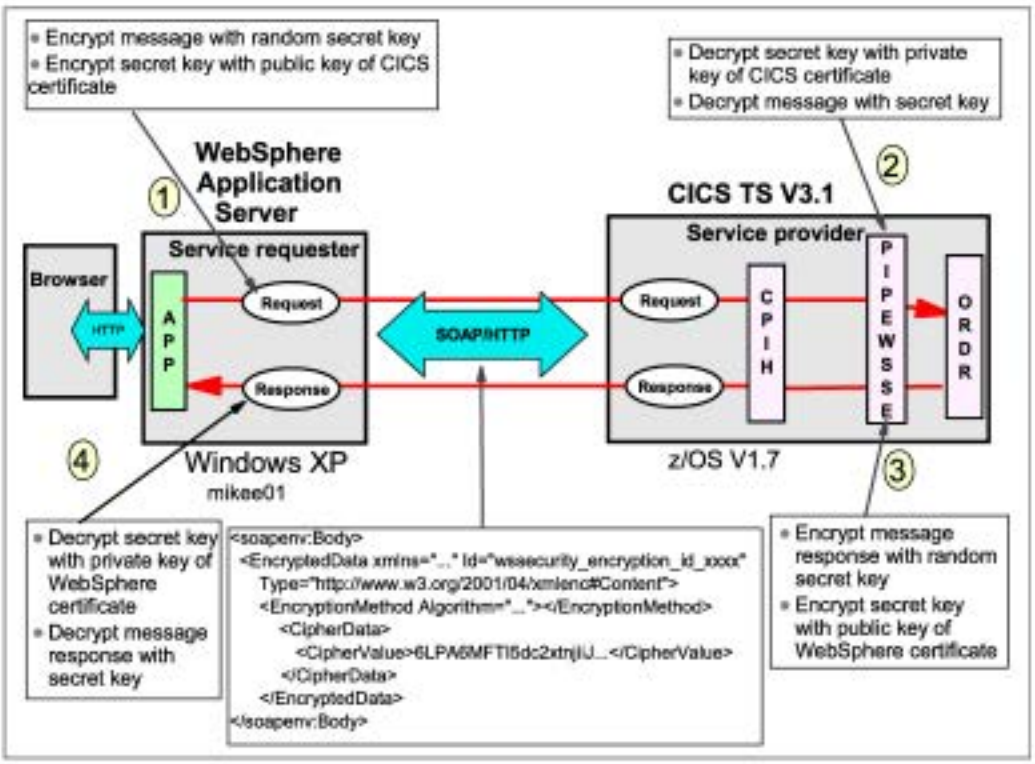


Figure 9-18 Enabling CICS WS-Security support for signatures

1. The service requester (WebSphere in our example) generates a BinarySecurityToken element from the WebSphere X.509 certificate. It then signs the message with its private key so the service provider (CICS in our example) knows that the message can only be sent by the WebSphere application server.
2. CICS validates the signature with the WebSphere public key. It also uses the owning user ID of the WebSphere X.509 certificate (WEBAS1) to run the ORDR transaction for the placeOrder service.
3. CICS then signs the response message with its private key so that WebSphere knows that the message can only be sent by the CICS region.
4. WebSphere validates the signature with the public key of CICS.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 360/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000360

Important: There is a significant performance overhead when using WS-Security and XML digital signatures. See 7.5, "Comparison of transport versus SOAP message security" on page 241 for recommendations on choosing between WS-Security and SSL when implementing an integrity security solution.

We document the following procedures to enable support for XML signature processing:

- Configuring the service requester for signature processing
- Configuring CICS for signature processing
- Testing the signature test scenario

Note: In our signature test scenario we sign both the request and response messages. It is also possible to sign only the request, or to sign only the response.

9.5.1 Configuring the service requester for signature processing

In this section we show how to configure the client J2EE application to send and receive signed SOAP messages. At a high level, the steps to do this are:

- Import the CatalogSec2 WebSphere Application to the Application Server Toolkit (AST).
- Configure the request generator for signing.
- Configure the response consumer for signing.
- Re-deploy the Web Service client application.

In the following sub-sections we provide step-by-step details for each procedure.

Importing the base application

To configure our Web Service client to use WS-Security, we used the IBM WebSphere Application Server Toolkit V6.1 (AST).

See "Importing the base application" on page 305 for instructions on how to import the application. For the signature test scenario, we specified:

- The name of the workspace folder as `C:\AST\CatalogSec2\CatalogSec2Signing`
- The name of the EAR project as `CatalogSec2Sign`.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000361

Configuring the request generator for signing

Perform the following steps to configure the request generator for signing:

1. Expand the **Dynamic Web Project** (**CatalogSec2Web**) in the Project Explorer and open (double-click) the deployment descriptor as shown in Figure 9-19.

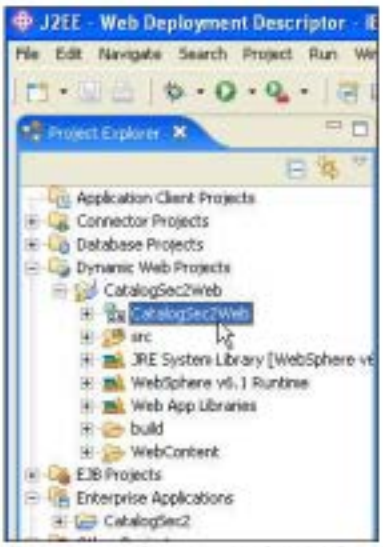


Figure 9-19 Opening the CatalogSec2Web Deployment Descriptor

2. Go to the WS Extension page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Request Generator Configuration**.
   c. Expand **Integrity**, click **Add**, and in the Integrity dialog box (Figure 9-20) enter the following data:
      - Enter a name identifying the part, for example, `int_body`.
      - Select the order in which the signature is generated. Multiple integrity parts can be specified. In our case, we selected 1.
      - Click **Add** for the Message Parts, and one integrity part is created. The default created part is the SOAP body. We accepted the default.
   d. Click **OK** and save the configuration by pressing Ctrl+s.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 362/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000362

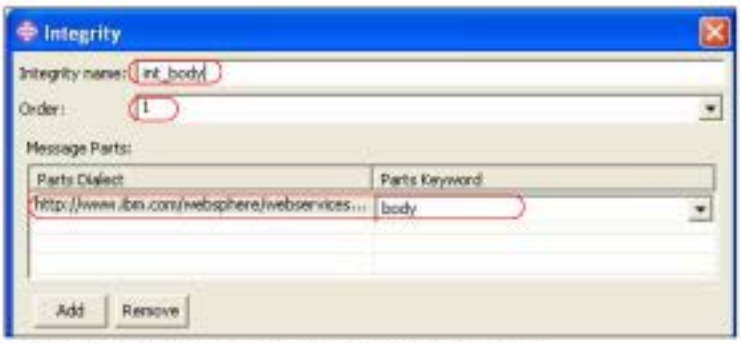


Figure 9-20 Integrity dialog in the WS Extensions page

3. Open the WS Binding page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Security Request Generator Binding Configuration**.
   c. Expand **Token Generator** and click **Add.** In the Token Generator dialog box (Figure 9-21) enter the following data:
      - Enter a Token generator name, for example, `gen_dsigtgen`.
      - For the Token generator class, select the **X509TokenGenerator**.
      - Do not select a Security Token.
      - Select **Use value type**, and then select **X509 certificate token v3** and the X509CallbackHandler.
      - Select **Use key store** and make the appropriate entries. Ours were:

        Password: `itso`
        Path: `C:\SG247206\keystore\was.jks`
        Type: `JKS`

      - Click **Add** under Key and make the appropriate entries. Ours were:

        Alias: `wascert_rsa`
        Key password: `itso`
        Key name: `CN=wascert, O=IBM, C=US`

        Note: This is the key name of the WebSphere certificate whose private key is used to sign the request message.

      - Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 363/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000363

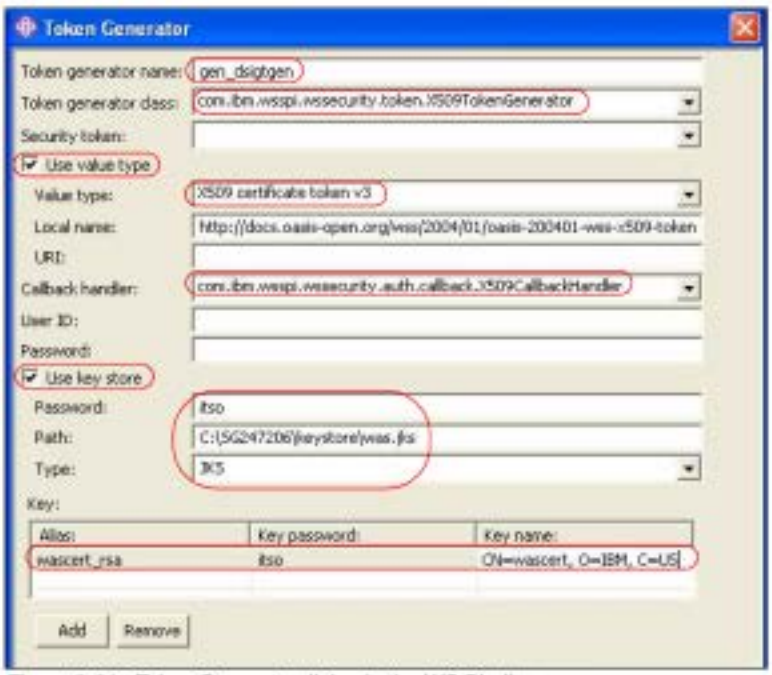


Figure 9-21 Token Generator dialog in the WS Binding page

4. Expand **Key Locators**, click **Add**, and in the Key Locator dialog box (Figure 9-22) enter the following data:

   a. Enter a Key locator name, for example, `gen_dsigklocator`.

   b. Select **KeyStoreKeyLocator** as the Key locator class.

   c. Select **Use key store** and make the appropriate entries. Ours were:

      Password: `itso`
      Path: `C:\SG247206\keystore\was.jks`
      Type: `JKS`

   d. Click **Add** under Key and make the appropriate entries. Ours were:

      Alias: `wascert_rsa`
      Key password: `itso`
      Key name: `CN=wascert, O=IBM, C=US`

   e. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
Online Notary.net
Page 364/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000364

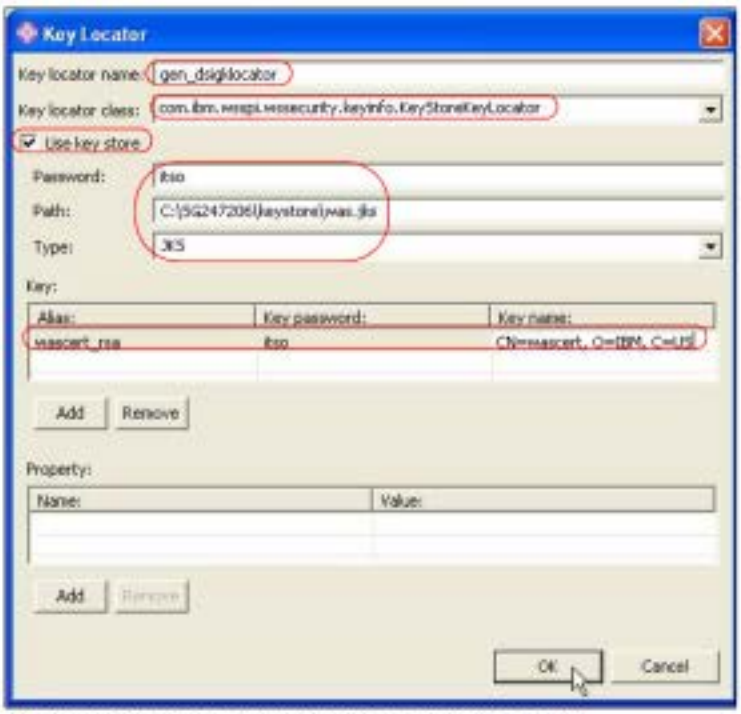


Figure 9-22 The Key Locator dialog in the WS Binding page

5. Expand **Key Information**, click **Add**, and in the Key Information dialog box (Figure 9-23) enter the following data:
   a. Enter a Key information name, for example, `gen_dsigkeyinfo`.
   b. Select **STRREF** (Direct reference) as the Key information type. The Key information class is filled automatically.

      The possible Key information types are:

      - STRREF Direct reference
      - EMB Embedded reference
      - KEYID Key identifier reference
      - KEYNAME Key name reference
      - X509ISSUER x.509 issuer and serial number reference

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 365/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000365

Note: We specify STRREF because we want WebSphere to send the public key as part of the complete certificate, since we will use the certificate for authentication in addition to decrypting the signed message.

It is also possible to configure an indirect reference to the public key, such as the key identifier or the key name. In this case, CICS uses the key reference to identify the RACF certificate that contains the public key.

c. Select **Use key locator** and select **gen_dsigklocator** from the Key locator drop-down list. Select the predefined Key name **CN=wascert, O=IBM, C=US**.

d. Select **Use token**, and select **gen_dsigtgen** from the drop-down list.

e. Click **OK**.

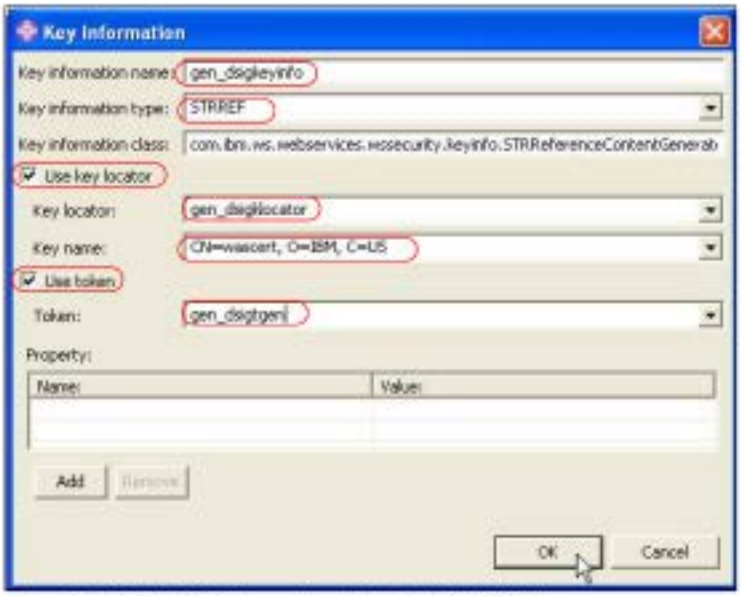


Figure 9-23 The Key Information dialog in the WS Binding page

6. Expand **Signing Information**, click **Add**, and in the Signing Information dialog box (Figure 9-24) enter the following data:

   a. Enter a Signing Information Name, for example, `sign_body`.

   b. Enter a Key information name, for example, `sign_kinfo`.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 366/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000366

c. Select the Key information element previously defined. In our case this was **gen_dsigkeyinfo**.

d. Click **OK**.

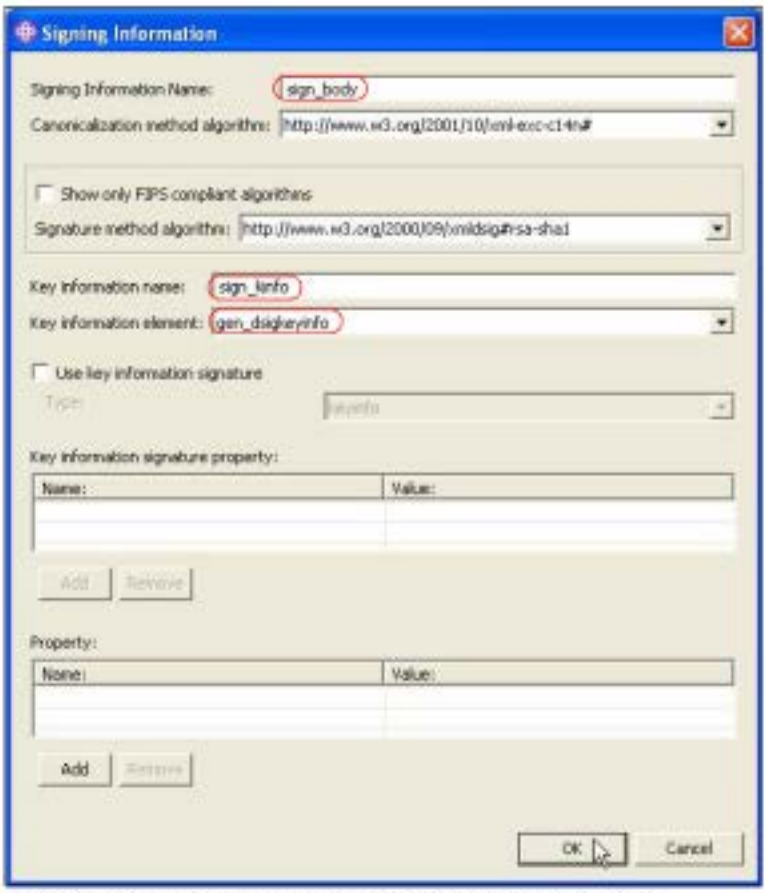


Figure 9-24 The Signing Information dialog in the WS Binding page

7. Expand **Part References**, click **Add**, and in the Part Reference dialog box (Figure 9-25) enter the following data:

   a. Enter a Part reference name, for example, `sign_part`.

   b. Select the Integrity part from the list of parts defined on the WS Extensions page. In our case, we select **int_body**.

   c. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 367/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000367



Figure 9-25 The Part Reference dialog in the WS Binding page

8. Expand **Transforms**, click **Add**, and in the Transform dialog box (Figure 9-26) enter the following data:
   a. Enter a name, for example, `sign_trans`.
   b. Click **OK**.



Figure 9-26 The Transform dialog in the WS Binding page

9. Save the configuration by pressing Ctrl+s.

Configure the response consumer for signing

In this section we show how to configure the client J2EE application in order to receive a signed SOAP message.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000368

Use the following steps to do this:

1. Expand the **Dynamic Web Project** (**CatalogSec2Web**) in the Project Explorer and open (double-click) the deployment descriptor.
2. Go to the WS Extension page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Response Consumer Configuration**.
   c. Expand **Required Integrity** and click **Add**. In the Required Integrity dialog box (Figure 9-27) enter the following data:
      - Enter a Required integrity name, for example, `reqint_body`.
      - Select the Usage type, either Required or Optional. If the usage type is Required, an unsigned response message throws a SOAP fault. If the usage type is Optional, an unsigned message is received. We selected **Required**.
      - Click **Add** for Message Parts, and one message part is created. The default created part is the SOAP body. We accepted the default.
   d. Click **OK** and save the configuration by pressing Ctrl+s.

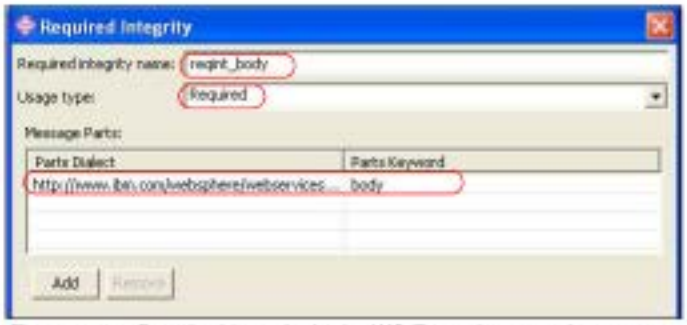


Figure 9-27 Required integrity in the WS Extension page for response consumer

3. Open the WS Binding page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Security Response Consumer Binding Configuration**.
   c. Expand **Token Consumer**, click **Add**, and in the Token Consumer dialog box (Figure 9-28) enter the following data:
      - Enter a Token consumer name, for example, `con_dsigtcon`.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000369

- Select **com.ibm.wsspi.wssecurity.token.X509TokenConsumer** as the Token consumer class.
- Select **Use value type**.
- Select **X509 certificate token**. The Local name is selected automatically.

> **Note:** It is important to specify X509 certificate token for the token value type. An error occurred in our test when we specified X509 certificate token v3.

- Select **Use jaas.config** and enter `system.wssecurity.X509BST` as jaas.config name.
- Select **Use certificate path settings**.
- Select **Trust any certificate**.

> **Important:** We specified **Trust any certificate** for our test. By selecting this option, the signature is validated by the certificate sent with the message without the certificate itself being validated.
>
> However, in a production environment you are unlikely to use this option because it allows any client with a valid XML digital signature certificate to have access to your WebSphere application server.
>
> WebSphere provides different ways of specifying trust for digital signature certificates.
>
> - If you use a self-signed certificate, you can specify the issuer name or serial number of trusted certificates in a jaas.config property.
> - For both self-signed and CA-signed certificates, you can specify a *trust anchor* and a *trusted certificate store* using a Certificate path reference.

d. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Page 370/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000370

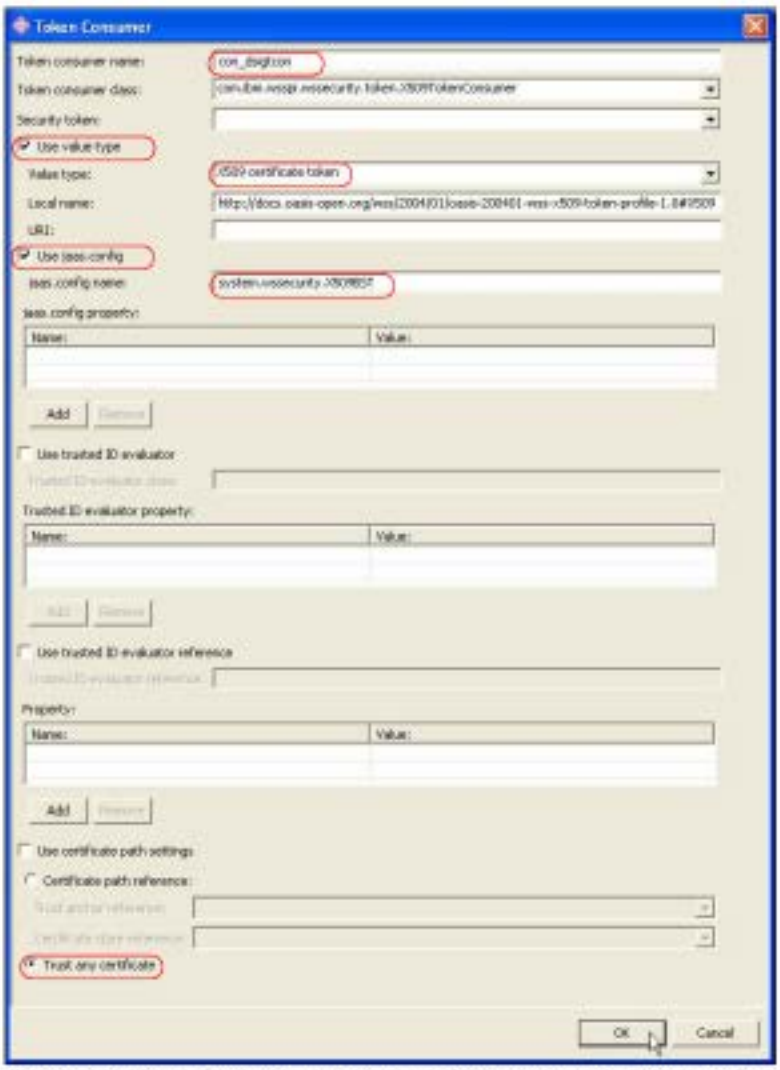


Figure 9-28 Token Consumer dialog in the WS Binding page for Response Consumer

4. Expand **Key Locators**, click **Add**, and in the Key Locator dialog box (Figure 9-29) enter the following data:
   a. Enter a Key locator name, for example, `con_dsigklocator`.
   b. Select **KeyStoreKeyLocator** as the Key locator class.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 371/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000371

c. Select Use key store and make the appropriate entries. Ours were:

Password:	itso
Path:	C:\SG247206\keystore\was.jks
Type:	JKS

d. Click **Add** under Key and make the appropriate entries. Ours were:

Alias:	cicscert
Key password:	itso
Key name:	CN=CICSCERT, O=ITSO, C=US

> **Note:** This is the key name of the CICS certificate. The private key of this certificate is used to sign the response message.

e. Click **OK**.

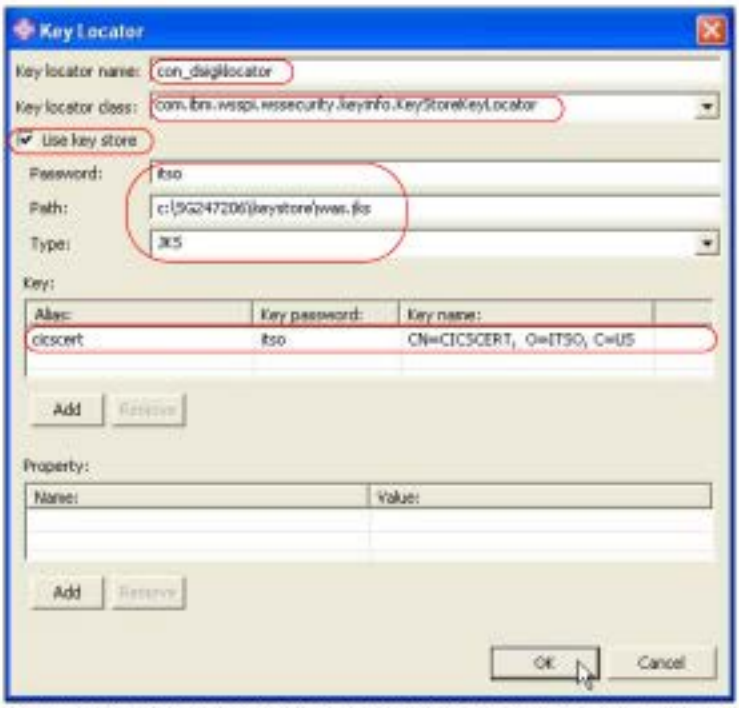


Figure 9-29 Key Locator dialog in the WS Binding page for response consumer

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 372/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000372

5. Expand **Key Information**, click **Add**, and in the Key Information dialog box (Figure 9-30) enter the following data:

   a. Enter a Key information name, for example, `con_dsigkeyinfo`.

   b. Select **STRREF** as the Key information type. The Key information class is filled automatically.

   > **Note:** We specify STRREF because CICS always sends the complete certificate when signing an outbound message; therefore, the WebSphere response consumer must be configured to expect the CICS certificate in the security header of the response message.

   c. Select **Use key locator** and select **con_dsigklocator** from the Key locator drop-down list. Select the predefined Key name as **CN=CICSCERT, O=ITSO, C=US**.

   d. Select **Use token** and select **con_dsigtcon** from the Token drop-down list.

   e. Click **OK**.

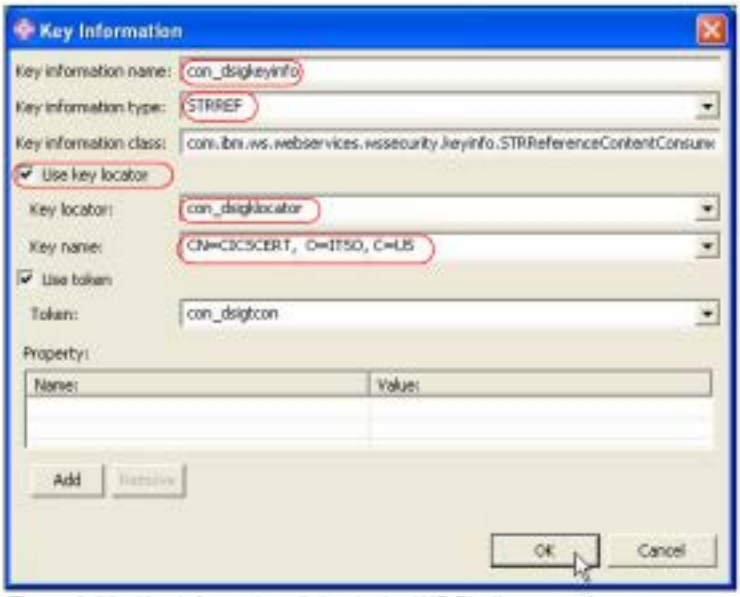


Figure 9-30 Key Information dialog in the WS Binding page for response consumer

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 373/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000373

6. Expand **Signing Information**, click **Add**, and in the Signing Information dialog box (Figure 9-31) enter the following data:
   a. Enter a Signing Information Name, for example, `sign_body`.
   b. Select **http://www.w3.org/2001/10/xml-exc-c14n#** as the Canonicalization method algorithm.
   c. Select **http://www.w3.org/2000/09/xmldsig#rsa-sha1** as the Signature method algorithm.
   d. Click **Add** in Signing key information.
      - Enter a Key information name, for example, `sign_kinfo`.
      - Select the **con_dsigkeyinfo** from the Key information element drop-down list.
   e. Click **OK**.

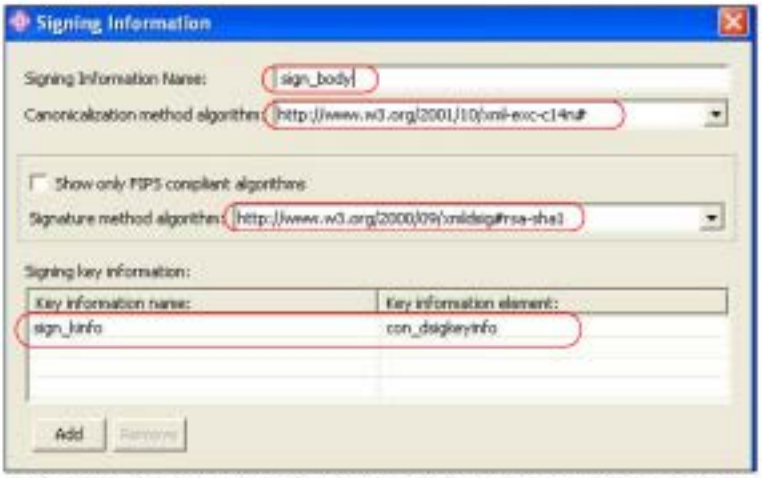


Figure 9-31 Signing Information dialog in the WS Binding page for response consumer

7. Expand **Part References**, click **Add**, and in the Part Reference dialog box (Figure 9-32) enter the following data:
   a. Enter a Part reference Name, for example, `sign_part`.
   b. Select **reqint_body** as the Required integrity.
   c. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 374/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000374



Figure 9-32 The Part Reference dialog in the WS Binding page for response consumer

8. Expand **Transforms**, click **Add**, and in the Transform dialog box (Figure 9-33) enter the following data:
   a. Enter a Name, for example, `sign_trans`.
   b. Select **http://www.w3.org/2001/10/xml-exc-c14n#** as the algorithm.
   c. Click **OK**.



Figure 9-33 The Transform dialog in the WS Binding page for response consumer

9. Save the configuration by pressing Ctrl+s.

Redeploying the Web service application

After configuring a signature security constraint for the service requester application, we exported a new EAR file called CatalogSec2_WS-Security_Signature.ear. We followed the same process that is

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 375/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000375

described in "Installing the service requester" on page 87 to deploy the CatalogSec2 WS-Security Signature.ear.

For further information on configuring WS-Security in WebSphere refer to *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257.

9.5.2 Configuring CICS for signature processing

In this section we show how to configure the CICS pipeline to receive and send signed SOAP messages, and to run the placeOrder request under the user ID associated with the WebSphere certificate contained in the request message.

Example 9-19 shows the pipeline configuration file that we used for the signature test scenario. It includes the message handler DFHWSSE1, and the configuration information for the handler.

Example 9-19 Pipeline config file, ITSO_7206_wssec_signature_provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/hts/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/hts/cics/pipeline
  provider.xsd ">
    <transport>
      <default_transport_handler_list>
          <handler>
             <program>CIKSMSGH</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
    </transport>
    <service>
      <service_handler_list>
        <handler>
          <program>SMI====6</program>
          <handler_parameter_list/>
        </handler>
        <wsse_handler>
          <dfhwsse_configuration version="1">
            <authentication mode="signature">
            <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
            </authentication>
            <expect_signed_body/>
            <sign_body>
              <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm>
              <certificate_label>CICSCERT</certificate_label>
            </sign_body>
          </dfhwsse_configuration>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000376

```
        </wsse_handler>
      </service_handler_list>
      </terminal_handler>
        <cics_soap_1.2_handler/>
      </terminal_handler>
    </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

The <wsse_handler> element shown in Example 9-19 contains a <dfhwsse_configuration> element that specifies configuration information for DFHWSSE1.

Important: If the <wsse-handler> element is included in a provider pipeline configuration, CICS processes signed SOAP messages by default.

- The <authentication mode="signature"> element specifies that inbound messages must contain an X.509 certificate in a BinarySecurityToken.

  If <authentication mode="signature"> is specified then the client X.509 certificate must be imported to RACF and attached to the CICS key ring since CICS runs the service request under the user ID associated with the client certificate.

  Note: CICS does not support certificate name filtering for signed SOAP messages.

  If <authentication mode="signature"> is not specified, and if the complete X.509 certificate is included in the request message, then the client X.509 certificate does not need to be imported to RACF since the public key required to validate the signature is extracted from the X.509 certificate contained in the SOAP message.

- <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm> is a child element of the *<authentication>* element that specifies the URI of the signature algorithm.

  Note: We found that it is necessary to specify a valid signature algorithm in service provider mode even if the algorithm used in the decryption of the signature is the one specified in the SOAP message rather than in the pipeline configuration file.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 377/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000377

- The <expect_signed_body/> element indicates that the *<body>* of the inbound message must be signed. If the body of an inbound message is not correctly signed, CICS rejects the message with a security fault.

  We specify <expect_signed_body/> because we want to prevent the placeOrder service being accessed unless the body of the SOAP message is signed.
- The <sign_body/> element indicates that the *<body>* of the outbound message must be signed, and provides information about how the message is to be signed.
- <algorithm>http://www.w3.org/2000/09/xmldsig#rsa-sha1</algorithm> is a child element of the *<sign-body>* element which specifies the URI of the signature algorithm to be used for signing the response message. This is the only signature algorithm supported by CICS for outbound messages.

  Attention: During our tests we encountered codepage issues when specifying the *<algorithm>* element. The configuration file should be coded in an English encoding such as EBCDIC-CP-US. When using a terminal emulation session with a non-English codepage, we noticed that the symbol '#' was a different hex value in the non-English codepage than in the EBCDIC-CP-US codepage (in which the symbol '#' is x'7B'). This resulted in error DFHPI0723 with the message:

  ```
  'The value for the algorithm specified for the authentication is
  not supported'.
  ```

  We recommend that you use a terminal emulation session with an English codepage when editing the pipeline configuration file.
- <certificate_label>CICSCERT</certificate_label> is a child element of the <sign-body> element that specifies the label associated with the CICS certificate installed in RACF. This certificate contains the PKDS key label of the private key that is used to sign the message response. The public key associated with the private key is then sent in the SOAP response message, allowing the signature to be validated by WebSphere.

We completed the CICS configuration by changing the pipeline PIPEWSSE (used for the placeOrder service) to specify the pipeline configuration file that we have created for signature processing. We used the following CEDA command:

```
CEDA ALTER PIPELINE(PIPEWSSE) GROUP(S3C1EXWS)
Configfile(/CIWS/S3C1/config/ITSO_7206_wssec_signature_provider.xml)
```

We then re-installed the changed pipeline resource definition.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000378

9.5.3 Testing the signature scenario

An example of the signed request message sent by WebSphere is shown in Example 9-20.

Example 9-20 Signed SOAP request message

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse
      curity-secext-1.0.xsd">
      <wsse:BinarySecurityToken
        EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-s
        oap-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="X509Token_1"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wsse
        curity-utility-1.0.xsd">MIIJCQUCCAmgAwIBAgIEVbOdzDANBgkqhkiG9w0BAQUFADs
        ZNQswCQYDVQQGEwJVUzEUMBIGA1UEChMLRVF1aWZheCBTZWN1cmUxJjAkBgNVBAsTHUVx
        dWlmYXggU2VjdXJlIENlcnRpZmljYXRlIEF1dGhvcml0eTANBgkqhkiG9w0BAQUFAAOBgQ
        CSgYEAwkTaDEMMAoGA1UECxMDQ0RTMRswGQYDVQQDExJDeWJlcnRydXN0IEdsb2JhbCBS
        b290MB4XDTA0MDUxMzE4MjM0M1oXDTA5MDUxMzE4MjM0M1owRDELMAkGA1UEBhMCVVMx
        FjAUBgNVBAoTDUVudHJ1c3QubmV0MSIwIAYDVQQLExlFbnRydXN0IFB1YmxpYyBDZXJ0
        aWZpY2F0aW9uIEF1dGhvcml0eTCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAwqZB
        aMoGQpzybJ+P3vCwNRbLVK3TTbrduxQ7YuK1HVUCey2VVSebES2yDaTrCLJ6/E2YTpB7
        HEt6+jbczvdF6xt/drfs7JD1ZVHX9zXDQHkm1EcovSKGr1w4nbyDkhI4ZJ2dES1XjnY
        d1T2ouE2BcRMTveJqfMC3mkn+jcn0EcSZlFrCo1fVN2/WL1axQwSJDm6coJFCNL4VY2V
        AAsjFI4PHfi9A1LciGcQ4SAwIYPhuSs7ASIANRcqHk+G9w9B4QJHAQUHgUEPLSFGGVW4rua
        zRDhIcdXFHHA4b6OrzIwObn+nh7Jd/uRIIyYHh6fcw+k4b+9F9U/IIPzIbJzgnpcM/rqHKK
        PIcb/vIB/HUuX+qrcGN+c12XBfrLs1S2IUPU66+qQx/JA3ygSJCuYfzKr2CcCPLFQ+cMMk
        nuj+UUIcAoUp+GF4Q==
      </wsse:BinarySecurityToken>
      <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
        <ds:SignedInfo>
          <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-ex
            c-c14n#">
            <ec:InclusiveNamespaces PrefixList="xsi xsd soapenv soapenc wsse
              ds" xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" />
          </ds:CanonicalizationMethod>
          <ds:SignatureMethod Algorithm=
            "http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
          <ds:Reference URI="#wssecurity_signature_1c_0">
            <ds:Transforms>
              <ds:Transform
                Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                <ec:InclusiveNamespaces PrefixList="xsi xsd soapenv soapenc wsu
                  p036" xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#" />
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000379

```
            </ds:Transform>
          </ds:Transforms>
          <ds:DigestMethod Algorithm=
            "http://www.w3.org/2000/09/xmldsig#sha1" />
          <ds:DigestValue>fh93ysjsoaBCwZ9RQqH2mI+WdYQ=</ds:DigestValue>
        </ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>I53WN8zcbfRmnz3LwVf3vZeFFDppZWAkgBU9+Sfu/I7J63WB3LsN
        3fRngwt7qLaRM3qGzjpsMNCbrKssyOP+dr5yBwvUIz4FukxyJ5pASVot7qPRRfj9p60/Y
        01u26zON96Q5cljkqWpl6+tNZqHwLiNfpwQyJggII+8izQU8nkQ=
      </ds:SignatureValue>
      <ds:KeyInfo>
        <wsse:SecurityTokenReference>
          <wsse:Reference URI="#x509bst1 1"
            ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-ws
            s-x509-token-profile-1.0#X509v3" />
        </wsse:SecurityTokenReference>
      </ds:KeyInfo>
    </ds:Signature>
  </wsse:Security>
</soapenv:Header>
<soapenv:Body wsu:Id="wssecurity_signature_id_0"
  xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
  ty-utility-1.0.xsd">
  <p635:DFHQXCVM xmlns:p635="urn://www.DFHQXCVM.DFHQXCP5.Request.com">
    <p63b:ca_request_id>ECIOR3R</p63b:ca_request_id>
    <p63b:ca_return_code>0</p63b:ca_return_code>
    <p63b:ca_response_message>CWBQZM1Z</p63b:ca_response_message>
    <p635:ca_order_request>
      <p63b:ca_userid>USER</p63b:ca_userid>
      <p635:ca_charge_dept>ITSO</p635:ca_charge_dept>
      <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
      <p635:ca_quantity_req>1</p635:ca_quantity_req>
      <p635:filler1 xsi:nil="true" />
    </p635:ca_order_request>
  </p635:DFHQXCVM>
</soapenv:Body>
</soapenv:Envelope>
```

- The header includes the mustUnderstand="1" attribute, which indicates that either this header must be processed or a SOAP fault thrown.
- The <wsse:BinarySecurityToken> contains the base64binary encoding of the WebSphere certificate. This includes the public key that CICS uses to verify the signature.
- The <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" /> element specifies the algorithm used to sign the message digest

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000380

- The <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" /> element specifies the algorithm used to produce the message digest.
- The <ds:DigestValue> element contains the value of the message digest.
- The <ds:SignatureValue> element contains the value of the signed message digest. It is the digest value encrypted with the requester's private key.

Figure 9-34 shows the ORDR transaction running with the user ID WEBAS1 (the user ID associated with the WebSphere certificate), while the CPIH transaction continues to execute with the CICSNW user ID.

```
 INQUIRE TASK
 STATUS:  RESULTS - OVERTYPE TO MODIFY
  Tas(CCCCCE5) Tra(CPIH)          Sus Tas Pri( 001 )
     Sta( ) Use(CICSNW  ) Ubn(B73C2E9E2114284E) Hty(RZCBNOT2)
  Tas(CCCCCE6) Tra(ORDR)          Sus Tas Pri( 001 )
     Sta( ) Use(WEBAS1  ) Ubn(B73C2E9E24801B9B) Hty(ECI     )
  Tas(CCCCCE8) Tra(CEMT) Fac(E024) Sus Ter Pri( 000 )
     Sta(SD) Use(NIGELW  ) Ubn(B73C2E9E2B044149) Hty(ZCIOWAIT)
  Tas(CCCCCE9) Tra(CEMT) Fac(E024) Sus Ter Pri( 255 )
     Sta(TO) Use(NIGELW  ) Ubn(B-3C25A641E9CCCE1
                                                    SrSID=SIDJ APPLID=A6P0S3CL
```

Figure 9-34 ORDR executing with user ID associated with WebSphere certificate

The corresponding signed response message sent by CICS is shown in Example 9-21.

Example 9-21 Signed SOAP response message

```
<?xml version="1.0" encoding="UTF8" standalone="no" ?>
 <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <SOAP-ENV:Header>
     <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-
        200401-wss-wssecurity-secext-1.0.xsd" SOAP-ENV:mustUnderstand="1"
       xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
       xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
       wssecurity-utility-1.0.xsd"
       xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
       <wsse:BinarySecurityToken
         EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss
         -soap-message-security-1.0#Base64Binary"
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000381

```
  ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x5
  09-token-profile-1.0#X509"
  wsu:Id="x509cert-CC0"MIIC4DCCAkigAwIBAgIEAOAVBjAKBggqhkjOPQQDBAUA
    QYDVQQGEwJVSzEPMA0GA1UECBMGTWVYIFFlcsgxETAPBgNVBAcTCEVYCRETDTELMAkG
    CwYDVQQKEwRJVFNMMQ0wCwYDVQQLEwRNQUNFMRIwEAYDVQQDEwlNQUNFIFFuQCBFeCAx
    FVREaMBYDVQQDEw9DSUNTQ3VSVDAeFw0wNDA4wNDBaFw0xNTAxMDEwMzU5WjBT
    IaMBgxCzAJBgNVBAcTAlVIMREwDwYDVQQIEwhOb3gwMJBSSzERMA8GA1UEBxMIRUBF
    UMPVFQxDTALBgNVBAoTBElJU09xDTALBgNVBAsTBFBTJDYxEjAQBgNVBAMTCUCBRElJg
    YVGqYElLIELxc1A/sqNVBMIIBMjCBlwYJKoZIhvcNAQMBMIGJAoGBAOLAAAGMAJC
    BiQKUgCCsEP+avGQtIkvbPJ+qNaUFF/9thk2aAJLr3Pup4KJvcA/JueLtn6sLQyQJVF
    snZgwQIIk97ELCCs755jwYdZcGSR2TgHxswdGrt4rt8EPgwcM3mSI6C9uWrn+Tvq1Jz
    NCDXpivdcRCTwRcQyeNUrJDKJwvdSr+bCwGr64LSdn2pQIDAQABuJIwcDARJ6gjgkgB
    hvhCAQQEMiPwRZwuZ2VuZGVkIGJ5IHRoZSBTZWNjdmIJe5BTZXJ2ZXIgZnQgZnJuN
    gKFJRQOYpPA4GAJL+JwEB/wQE4wIEUDAdRghWHQ4EFgQJ62+x+0kFhSysr93LO6Zk5n
    yYaPgwRCYJKc7/+wcN4QFFRJ4NgYFATtVWjr/2p/+471F37RaTpUne7xq7eMn7.QC5ID
    u37ya+qC AxbfP574/Kgqn+egYyYXRY14mCsrpRq3MHRN/nFIvw1FUNprnCe28F+1Cq
    3p3rzn+hfuRY4/Ip4t4CK9rUPIIjsvgxpuIrWfcM+/JrxYXcY3YIh4L/RBBs/CQ17xJ
</wsse:BinarySecurityToken>
<ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
    xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
    wssecurity-utility-1.0.xsd"
    xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
      <c14n:InclusiveNamespaces
        xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"
        PrefixList="ds wsu xenc SOAP-ENV soapenv soapenv xsd xsi "/>
    </ds:CanonicalizationMethod>
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
    <ds:Reference URI="#TheBody">
      <ds:Transforms>
        <ds:Transform
          Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
          <c14n:InclusiveNamespaces
            xmlns:c14n="http://www.w3.org/2001/10/xml-exc-c14n#"
            PrefixList="wsu SOAP-ENV soapenv soapenv xsd xsi "/>
        </ds:Transform>
      </ds:Transforms>
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
      <ds:DigestValue>eLbsd15PsgrzG25bDQdtyBoHDW0=</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
```


Document Id: [illegible]
[illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000382

```
          <ds:SignatureValue>ZyxxxQ/n7yDaGDYwsIIS2bMIbOdMWNruFJ/tT5HuWi0Ob4N7k5
            DFccH27wQbluEVFjkMjkKzOnOLHNgIzGqaBU4cV3hu6VDKr4QgBCmxtO6yDpyOYYC/e
            5rtjfCQUycgJlJYWdqd+ERN9hwbXX3wZZX3PY25W5QsDwH/aJ0e5HE=
          </ds:SignatureValue>
          <ds:KeyInfo>
            <wsse:SecurityTokenReference><wsse:Reference URI="#X509certOO"
              ValueType "http://docs.oasis-open.org/wss/2004/01/oasis-200401-
              wss x509 token profile 1.0#X509"/>
            </wsse:Security okenReference>
          </ds:KeyInfo>
        </ds:Signature>
      </wsse:Security>
    </SOAP-ENV:Header>
    <SOAP-ENV:Body
      xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
      wssecurity-utility-1.0.xsd" wsu:Id="TheBody">
      <DFHDAIC4HResponse
        xmlns "http://www.DFHDAIC4H.DFHWSCFN.Response.com">
        <ca_request_id>CIC4</ca_request_id>
        <ca_return_code>0</ca_return_code>
        <ca_response_message>ORDER SUCCESSFULLY PLACED
        </ca_response_message>
        <ca_order_request>
          <ca_userid>luisl106</ca_userid>
          <ca_charge_dept>tso    </ca_charge_dept>
          <ca_item_ref_number>10</ca_item_ref_number>
          <ca_quantity_req>1</ca_quantity_req>
          <filler1>...</filler1>
        </ca_order_request>
      </DFHDAIC4HResponse>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

The <wsse:BinarySecurityToken> contains the base64binary encoding of the CICS certificate. This includes the public key that WebSphere uses to verify the signature. The rest of the SOAP headers in the response message are similar to those in the request message.

SOAP fault messages

In this section we show some of the SOAP faults that you may see when testing signature processing.

- Figure 9-35 shows the browser response when CICS is expecting a signed message but the service requester does not sign the request.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000383

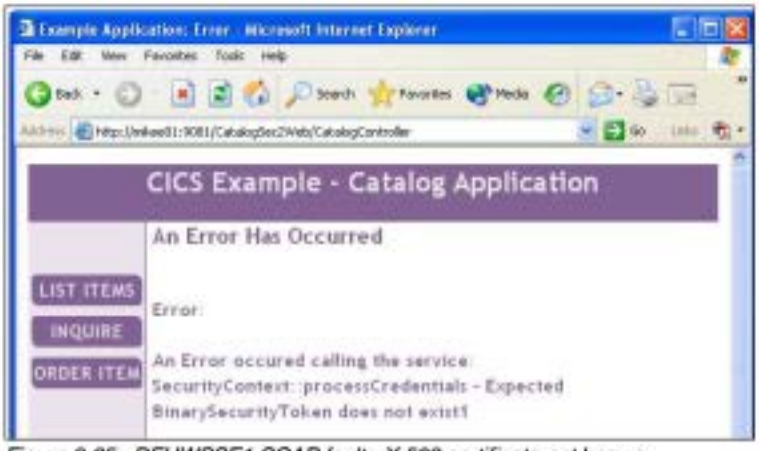


Figure 9-35 DFHWSSE1 SOAP fault - X.509 certificate not known

The SOAP fault for this message, shown in Example 9-22, highlights that CICS is expecting a BinarySecurityToken, which is not sent by WebSphere.

Example 9-22 DFHWSSE1 SOAP fault - X.509 certificate not known

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
    <faultcode>wsse:InvalidSecurity</faultcode>
    <faultstring>ERROR: Caught "XSECException" during operation:
      processMessage()</faultstring>
    <detail>
      <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
        <message>SecurityContext::processCredentials - Expected
          BinarySecurityToken does not exist</message>
        <errorcode>1</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 384/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000384

- Figure 9-36 shows the browser response when the client certificate is not associated to a user ID.

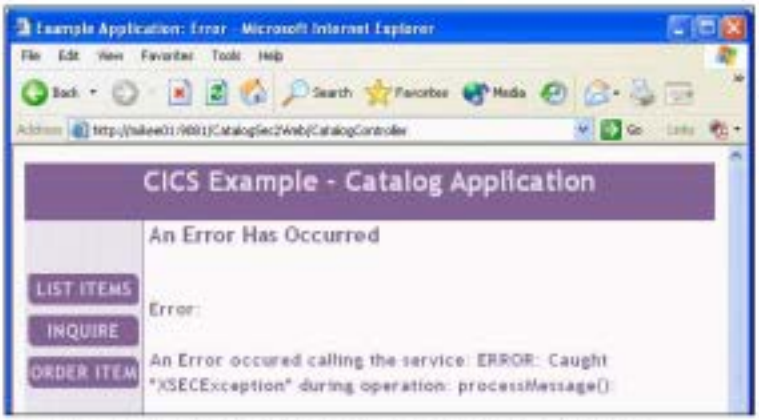


Figure 9-36 DFHWSSE1 SOAP fault - X.509 certificate not known

The SOAP fault for this message, shown in Example 9-23, highlights that there is no RACF user ID defined for the WebSphere certificate.

Example 9-23 DFHWSSE1 SOAP fault - X.509 certificate not known

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>wsse:FailedAuthentication</faultcode>
      <faultstring>ERROR: Caught *XSECException* during operation:
        processMessage()</faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
          <message>XSECKeyInfoResolverZos::extractUserId - Either no RACF user
            ID is defined for this certificate, or the certificate status is
            NOTRUST.</message>
        <errorcode>1</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 385/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000385

9.6 Encrypting a SOAP message

To provide confidentiality we can encrypt the contents of a SOAP message using XML encryption. If the SOAP message is encrypted, only a service consumer that knows the key for confidentiality can decrypt and read the message.

CICS supports XML encryption in both service provider and service requester modes.

In this section we show how a SOAP message can be encrypted by WebSphere Application Server and how the SOAP message can be decrypted by CICS. We also show how the response message can be encrypted by CICS and decrypted by WebSphere. This scenario is shown in Figure 9-37.

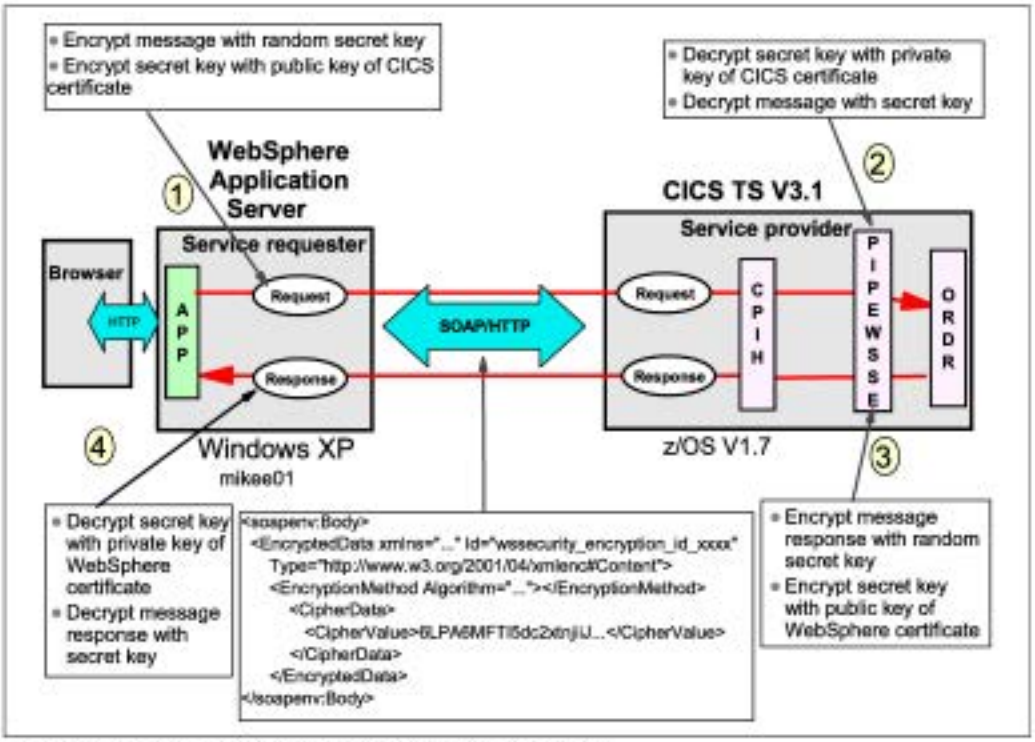


Figure 9-37 Enabling CICS WS-Security support for encryption

1. The WebSphere application server generates a random secret key and uses it to encrypt the SOAP message body. It also encrypts the secret key with the

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 386/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000386

CICS public key so only the CICS region can decrypt it. Both the encrypted data and the encrypted secret key are attached to the SOAP message.

2. CICS decrypts the secret key with its private key and uses it to decrypt the SOAP body.
3. CICS then encrypts the SOAP body of the response message with a different randomly generated secret key. It also encrypts the secret key with the public key of WebSphere so only the WebSphere application server can decrypt it.
4. The WebSphere application server decrypts the secret key with its private key and uses it to decrypt the SOAP body of the message response.

Recommendation: In our encryption scenario we use the same key pairs that we used in the signature scenario. In a production environment, however, we recommend that you use different key pairs for signature and encryption processing.

Important: There is a significant performance overhead when using WS-Security and XML encryption. See "Comparison of transport versus SOAP message security" on page 241 for recommendations on choosing between WS-Security and SSL when implementing a confidentiality security solution.

At a high level, the steps to enable support for XML encryption processing are the following:

- Configure the service requester for encryption.
- Configure CICS for encryption.
- Test the encryption test scenario.

Step-by-step details to accomplish these tasks are presented in the following sections.

Note: In our encryption test scenario we encrypt both the request and response messages. It is also possible to encrypt only the request, or to encrypt only the response. Equally, it is possible to both encrypt *and* sign a message request or response.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000387

9.6.1 Configuring the service requester for encryption

In this section we show how to configure the client J2EE application in order to send an encrypted SOAP message. At a high level the steps are:

- Import the Web service client application into AST.
- Configure the request generator for encryption
- Configure the response consumer for encryption.
- Re-deploy the Web Service client application

Import the Web service client application into AST

To configure our J2EE application for encryption, we used the IBM WebSphere Application Server Toolkit V6.1 (AST).

See "Importing the base application" on page 305 for instructions on how to import the application. For the signature test scenario, we specified

- The name of the workspace folder as:
  C:\AST\CatalogSec2\CatalogSec2Encrypt
- The name of the EAR project as:
  CatalogSec2Encrypt

Configure the request generator for encryption

Perform the following steps to configure the request generator for encryption:

1. Expand the **Dynamic Web Project** (**CatalogSec2Web**) in the Project Explorer and open (double-click) the deployment descriptor as shown in Figure 9-38

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000388

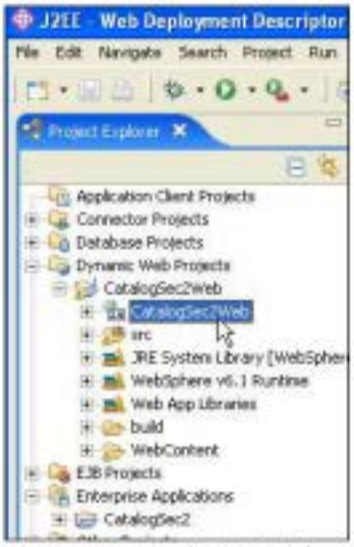


Figure 9-38 Opening the CatalogSec2Web Deployment Descriptor

2. Go to the WS Extension page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Request Generator Configuration**.
   c. Expand **Confidentiality** and click **Add**. In the Confidentiality dialog box (Figure 9-39) enter the following data:
      - Enter a Confidentiality name, for example, `conf_body`.
      - Select the order in which the encryption is generated. Multiple confidentiality parts can be specified, and you have to specify the order of generating the encryption. In our case, we select **1**.

      Note: The WS-Security runtime of WebSphere V6.1 supports multiple signature and encryption parts in one SOAP message. For multiple signature and encryption parts, you need to specify the processing order. For example, if you want to sign *and* encrypt the SOAP body, you should specify 1 in the Integrity dialog and 2 in the Confidentiality dialog.

      - Click **Add** for Message Parts, and one confidentiality part is created. The default created part is the SOAP body. We accepted the default.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 389/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000389

d. Click OK and save the configuration by pressing Ctrl+s.

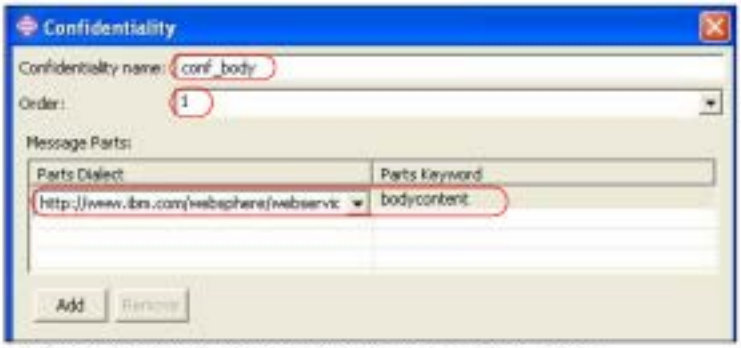


Figure 9-39 Confidentiality dialog in the WS Extensions page

3. Open the *WS Binding* page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Security Request Generator Binding Configuration**.
   c. Expand **Key Locators** and click **Add**. In the Key Locator dialog box (Figure 9-40) enter the following data:
      - Enter a Key locator name, for example, `gen_encklocator`.
      - Select **KeyStoreKeyLocator** as the Key locator class.
   d. Select **Use key store**. For key-related information refer to "Importing the base application" on page 332. Make the appropriate entries. In our case the values were:

      Password: `itso`
      Path: `C:\SG247206\keystore\was.jks`
      Type: `JKS`

   e. Click **Add** under Key and make the appropriate entries. Ours were:

      Alias: `CICSCERT`
      Key password: `itso`
      Key name: `CN=CICSCERT, O=ITSO, C=US`

      Note: This is the key name of the CICS certificate. The public key of this certificate is used by WebSphere to encrypt the request message.

   f. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000390

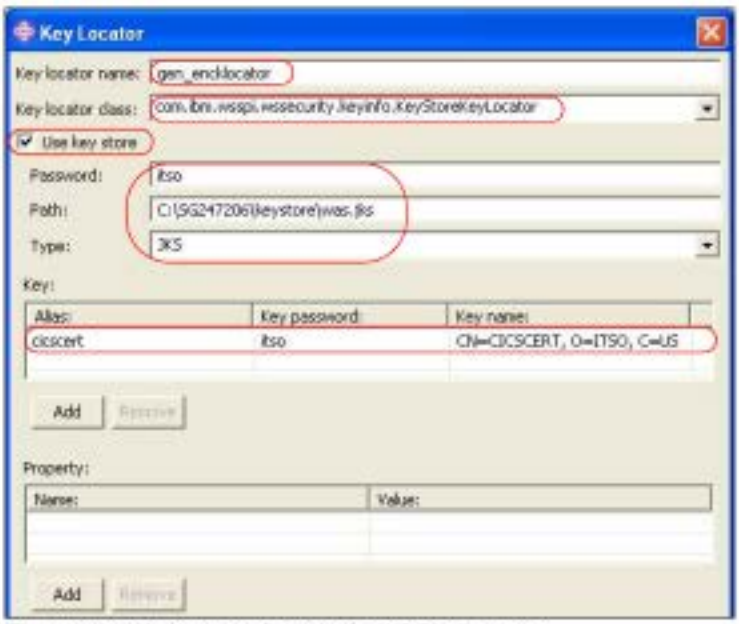


Figure 9-40 Key Locator dialog in the WS Binding page

4. Expand **Key Information** and click **Add**. In the Key Information dialog box (Figure 9-41) enter the following data:

   a. Enter a Key information name, for example, `gen_enckeyinfo`.

   b. Select **KEYNAME** as the Key information type. The Key information class is filled automatically. The Key information types are:

      - STRREF Direct reference
      - EMB Embedded reference
      - KEYID Key identifier reference
      - KEYNAME Key name reference
      - X509ISSUER x.509 issuer and serial number reference

      We specify KEYNAME because we want WebSphere to send the key information using the distinguished name of the CICS certificate.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 391/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000391

Recommendation: Use of KEYNAME for the key information type allows CICS to reference the RACF certificate directly because each certificate has a unique distinguished name and RACF provides an API for accessing the certificate using the distinguished name.

c. Select **Use key locator** and select **gen_enckIocator** from the Key locator drop-down list. Select the predefined Key name **CN=CICSCERT, O=ITSO, C=US**.

d. Click **OK**.

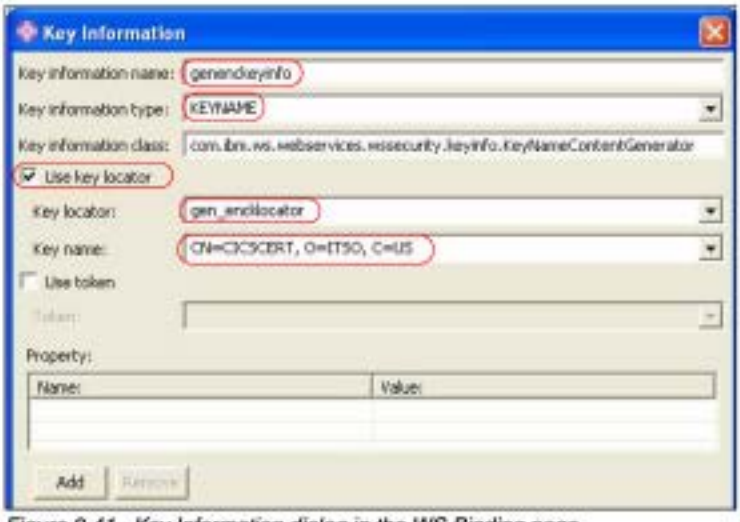


Figure 9-41 Key Information dialog in the WS Binding page

5. Expand **Encryption Information** and click **Add**. In the Encryption Information dialog box (Figure 9-42) enter the following data:

   a. Enter an Encryption Name, for example, `enc_body`.

   b. Enter a Key information name, for example, `enc_keyinfo`.

   c. Select **gen_enckey** as a Key information element.

   d. Select **conf_body** as a Confidentiality part.

   e. Allow the encryption algorithms to default.

   f. Click **OK**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 382/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000392

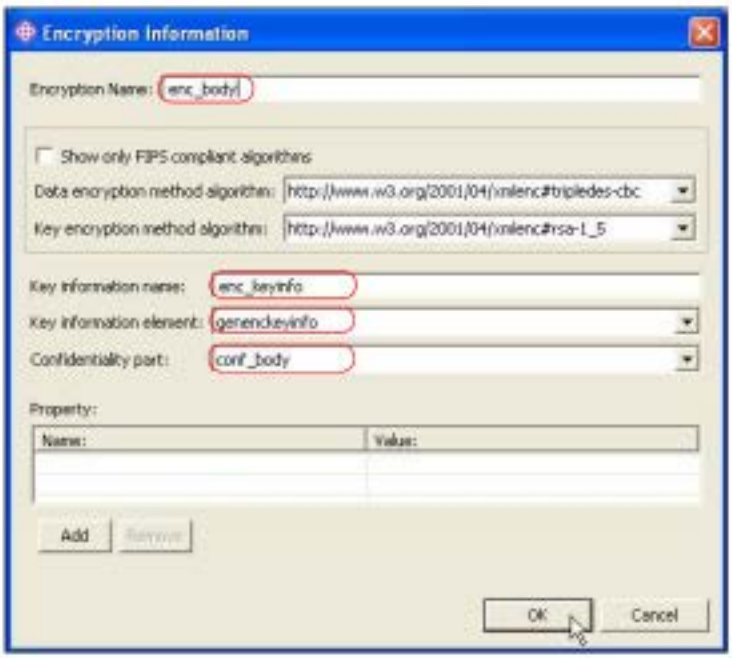


Figure 9-42 The Encryption Information dialog in the WS Binding page

6. Save the configuration by pressing Ctrl+s.

Configure the response consumer for encryption

In this section we show how to configure the client J2EE application in order to receive an encrypted SOAP message. To do this, perform the following steps:

1. Expand the **Dynamic Web Project** (**CatalogSec2Web**) in the Project Explorer and open (double-click) the deployment descriptor.
2. Go to the WS Extension page.
   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).
   b. Expand the **Response Consumer Configuration**.
   c. Expand **Required Confidentiality** and click **Add**. In the Required Confidentiality dialog box (Figure 9-43) enter the following data:
      - Enter a Required confidentiality name, for example, `reqconf_body`.
      - Select the Usage type, either Required or Optional. If the usage type is Required, an unencrypted request message throws a SOAP fault. If the

Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000393

usage type is Optional, an unencrypted message is received. Select **Required**.

d. Click **Add** for Message Parts, and one message part is created. The default created part is the SOAP body. We accepted the default.

e. Click **OK** and save the configuration by pressing Ctrl+s.

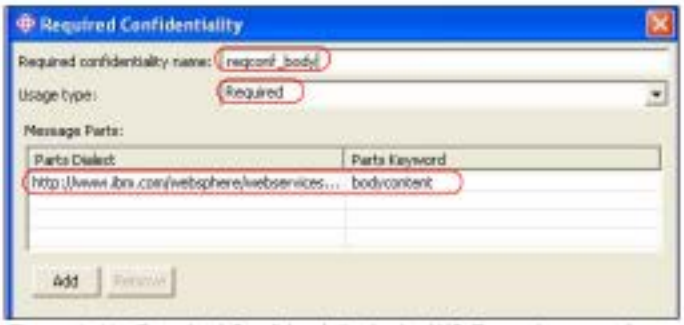


Figure 9-43 Required Confidentiality in the WS Extension page for response consumer

3. Open the WS Binding page.

   a. Select the **service/DFH0XCMNService3** reference (for our Order service) and **DFH0XCMNPortPO** (the port binding for the service reference).

   b. Expand the **Security Response Consumer Binding Configuration**.

   c. Expand **Token Consumer** and click **Add**. In the Token Consumer dialog box (Figure 9-44) enter the following data:

      - Enter a Token consumer name, for example, `con_enctcon`.
      - Select **com.ibm.wsspi.wssecurity.token.X509TokenConsumer** as the Token consumer class.
      - Select **Use value type**.
      - Select **X509 certificate token**. The Local name is selected automatically.

      Note: It is important to specify X509 certificate token for the token value type. An error occurred in our test when we specified X509 certificate token v3.

      - Select **Use jaas.config** and enter `system.wssecurity.X509BST` as the jaas.config name.
      - Select **Trust any certificate**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 394/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000394

d. Click **OK**.

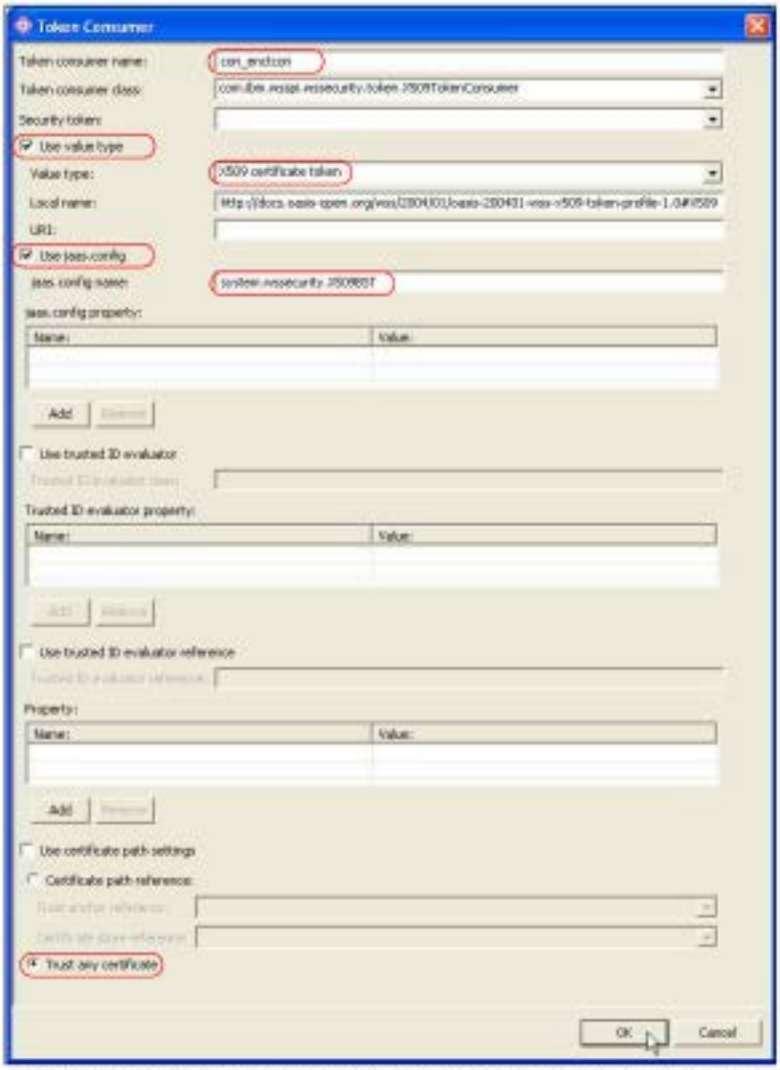


Figure 9-44 Token Consumer dialog in the WS Binding page for response consumer

4. Expand **Key Locators** and click **Add**. In the Key Locator dialog box (Figure 9-45) enter the following data:

   a. Enter a Key locator name, for example, `con_encklocator`.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000395

b. Select **KeyStoreKeyLocator** as the Key locator class.

c. Select **Use key store** and make the appropriate entries. Ours were:

Password:	`itso`
Path:	`C:\SG247206\keystore\was.jks`
Type:	`JKS`

d. Click **Add** under Key and make the appropriate entries. Ours were:

Alias:	`wascert_rsa`
Key password:	`itso`
Key name:	`CN=wascert, O=IBM, C=US`

> **Note:** This is the key name of the WebSphere certificate. The public key of this certificate is used by CICS to encrypt the response message.

e. Click **OK**.

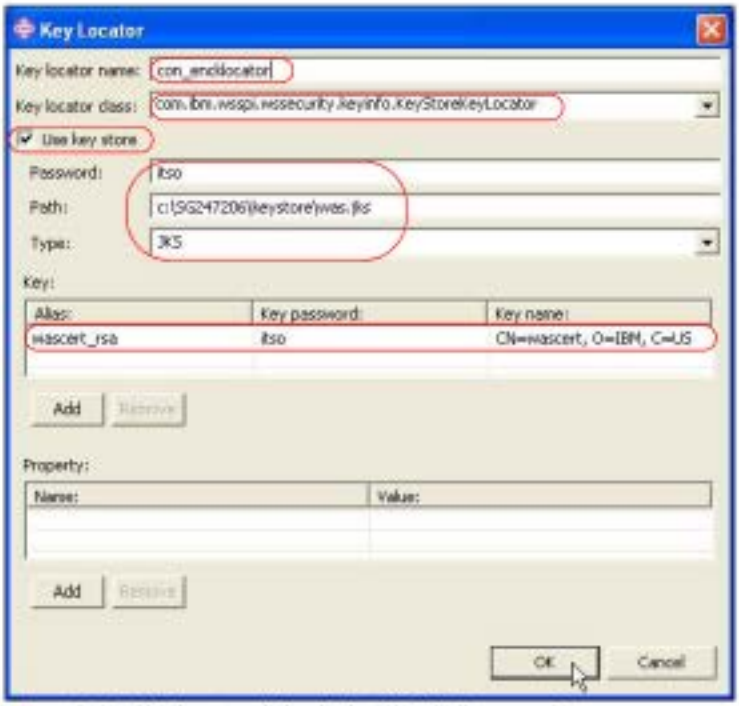


Figure 9-45 Key Locator dialog in the WS Binding page for response consumer

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
Online4victory.net
Page 396/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000396

5. Expand **Key Information** and click **Add**. In the Key Information dialog box (Figure 9-46) enter the following data:

   a. Enter a Key information name, for example, `con_enckeyinfo`.

   b. Select **STRREF** as the Key information type. The Key information class is filled automatically.

      We specify STRREF because CICS always sends the complete certificate when encrypting an outbound message. Therefore, the WebSphere response consumer must be configured to expect the WebSphere certificate in the security header of the response message.

   c. Select **Use key locator** and select **con_enclocator** from the Key locator drop-down list. Select the predefined Key name **CN=wascert, O=IBM, C=US**.

   d. Select **Use token** and select **con_enctcon** from the Token drop-down list.

   e. Click **OK**.

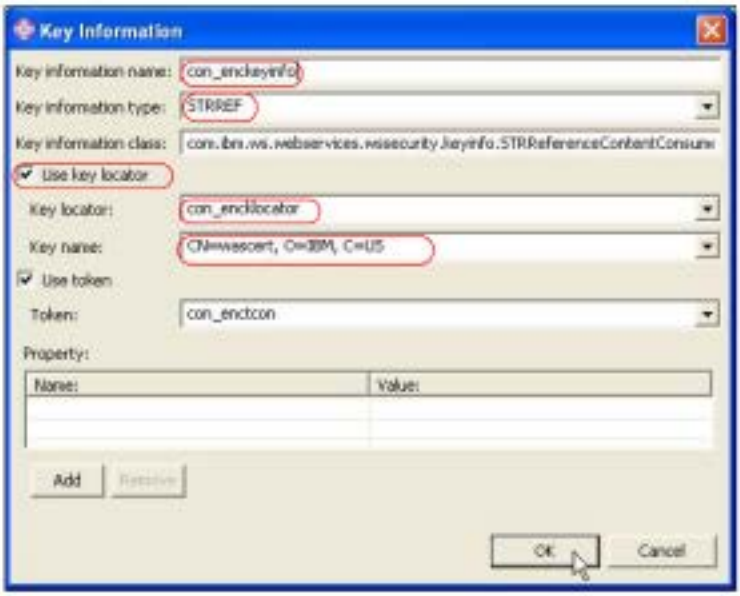


Figure 9-46 Key Information dialog in the WS Binding page for response consumer

6. Expand **Encryption Information** and click **Add**. In the Encryption Information dialog box (Figure 9-47) enter the following data:

   a. Enter a Encryption Name, for example, `enc_body`.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000397

b. Click **Add** in Encryption key information and make the appropriate entries:
   - Enter a Key information name, for example, `enc_kinfo`.
   - Select the **con_enckeyinfo** from the Key information element drop-down list.

c. Select **reqconf_body** as a Required Confidentiality part.

d. Allow the encryption algorithms to default.

e. Click **OK**.

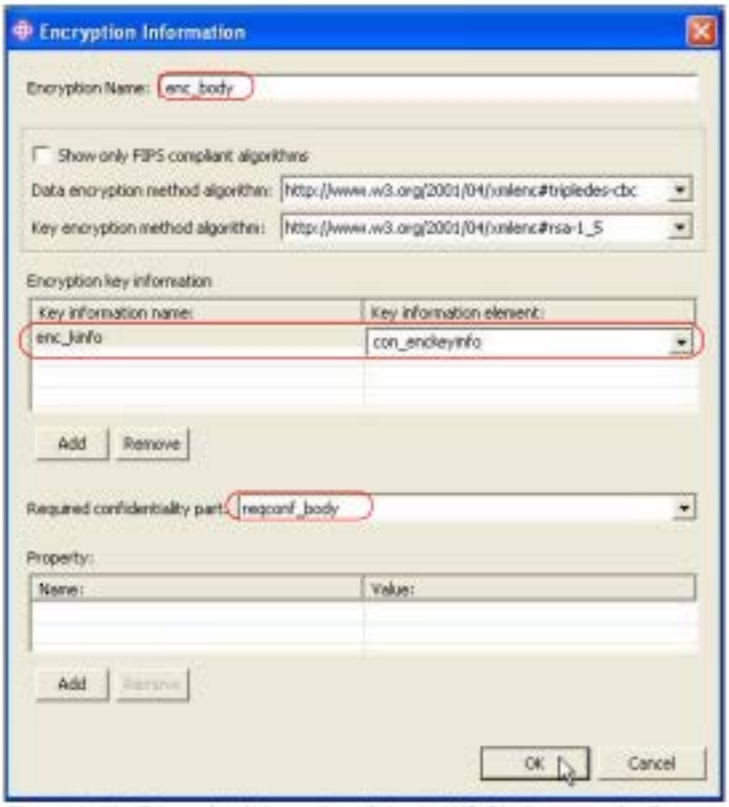


Figure 9-47 Encryption Information dialog in WS Binding page for Response Consumer

7. Save the configuration by pressing Ctrl+s.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 398/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000398

Redeploying the Web service application

After configuring an encryption security constraint for the service requester application, we exported a new EAR file called CatalogSec2_WS-Security_Encryption.ear. We followed the same process that is described in "Installing the service requester" on page 87 to deploy the CatalogSec2_WS-Security_Encryption.ear.

For further information on configuring WS-Security in WebSphere refer to *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257.

9.6.2 Configuring CICS for encryption

In this section we show how to configure the CICS pipeline to receive and send encrypted SOAP messages.

Example 9-24 shows the pipeline configuration file that we used for the encryption test scenario. It includes the message handler DFHWSSE1, and the configuration information for the handler.

Example 9-24 Pipeline config file, ITSO_7206_wsse_signature_provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
    <transport>
      <default_transport_handler_list>
          <handler>
             <program>CHMSHSG</program>
             <handler_parameter_list/>
          </handler>
      </default_transport_handler_list>
    </transport>
    <service>
      <service_handler_list>
        <handler>
          <program>SH[????]</program>
          <handler_parameter_list/>
        </handler>
        <wsse_handler>
          <dfhwsse_configuration version="1">
            <expect_encrypted_body/>
            <encrypt_body>
              <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc
              </algorithm>
              <certificate_label>WASCERT</certificate_label>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000399

```
            </encrypt_body>
          </dfhwsse_configuration>
        </wsse_handler>
      </service_handler_list>
      </terminal_handler>
        <cics_soap_1.2_handler/>
      </terminal_handler>
    </service>
  <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

- The <wsse_handler> element contains a <dfhwsse_configuration> element that specifies configuration information for DFHWSSE1.
- The <expect_encrypted_body/> element indicates that the <body> of the inbound message must be encrypted. If the body of an inbound message is not correctly encrypted, CICS rejects the message with a security fault.

  We specify <expect_encrypted_body/> because we want to prevent the placeOrder service being accessed unless the body of the SOAP message is encrypted.
- The <encrypt_body/> element indicates that the <body> of the outbound message will be encrypted, and provides information about how the message is to be encrypted.
- <algorithm>http://www.w3.org/2001/04/xmlenc#tripledes-cbc</algorithm> is a child element of the <encrypt-body> element that specifies the URI of the encryption algorithm to be used for encrypting the response message. We specified the triple-DES algorithm.

  Important: The encryption algorithm must match the data encryption method algorithm specified in the WebSphere application service response consumer configuration.

- <certificate_label>WASCERT</certificate_label> is a child element of the <encrypt-body> element that specifies the label associated with the WebSphere certificate installed in RACF. This certificate contains the public key that is used by CICS to encrypt the secret (or symmetric) key. The encrypted secret key is then sent in the SOAP message response.

We completed the CICS configuration by changing the pipeline PIPEWSSE (used for the placeOrder service) to specify the pipeline configuration file that we have created for encryption processing. We used the following CEDA command:

```
CEDA ALTER PIPELINE(PIPEWSSE) GROUP(S3C1EXWS)
Configfile(/CIWS/S3C1/config/ITSO_7206_wssec_encryption_provider.xml)
```


Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000400

We then re-installed the changed pipeline resource definition

9.6.3 Testing the encryption scenario

An example of an encrypted SOAP request message for the placeOrder service is shown in Example 9-25.

Example 9-25 Encrypted SOAP request message

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Header>
    <wsse:Security soapenv:mustUnderstand="1"
      xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
      wssecurity-secext-1.0.xsd">
      <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
        <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
        />
        <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
          <ds:KeyName>CN=CICSCERT, T=CIwss3c1-cert, OU=PSSC, O=ITSO,
            L=ENDICOTT, ST=NEW YORK, C=US</ds:KeyName>
        </ds:KeyInfo>
        <CipherData>
          <CipherValue>rh8=[y+]]]P4/g4ezibCPaxt=fxEZzQ3CSgQQ0USRMGqyEuy6b==sty9
            4CEuxDzdKV5JvSAsor8grDiExAlO2Ynn3]xc8#1Sk)1KKkV/DdiUS;C]xIyI*JK<=Da
            aT45ki/CChcruCr=9KJ2v5]oqaYAiPTHGr33C'dbcJGZjdPo=</CipherValue>
        </CipherData>
        <ReferenceList>
          <DataReference URI="#wssecurity_encryption_id_29" />
        </ReferenceList>
      </EncryptedKey>
    </wsse:Security>
  </soapenv:Header>
  <soapenv:Body>
    <EncryptedData Id="wssecurity_encryption_id_29"
      Type="http://www.w3.org/2001/04/xmlenc#Content"
      xmlns="http://www.w3.org/2001/04/xmlenc#">
      <EncryptionMethod
        Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
      <CipherData>
        <CipherValue>yJ+-MZ4:kuzjfpvdskqrMHQF9Fr+y9id'9skJnK;qby;t;6eNes6=nPbtJ
          /JvpGKsEwBchUwvtL..anRujc3W6;xcX/lG+x*gAq:zoCrykzh3/lbJeHqhp44sBk-Q1
          b1V/IeaJQ9b/-.Hq:JCa/wJWWwxxZnIkd;zZqBbwQV%.tJVNJCU;:P/wcxCt/rJ'W<rp
          1RsQb4DwuZcrHOf?<//*1FRfmyL9fDKR2flDn/QENSJAcxJLZb49u7vtQm%SvDOteF11%
          iDuu4CJC19S4CfENc3xxxidt2EZYGGK0z=s.8nWrmLPQa*eHxwGuQ6PyPCDs+xsPcO4
          n2/s43Hov4VDCYgvW2%0Q6OJ4aPc5cPuJci/cH2-xyz?b2v4Vlobe!IXX8sejbuFc5yW
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000401

```
            uJoCz6uwqDpbPltUTsGUxo36YtJuu9lQ7CTcuqurs25WLTSqdZLMeRyKIygpYueT+LIIp
            ihdNuopZbC9LPag3xncb+CAnx2xwIGYLtalEE4g8rOc4EU1IY5NHqqyCXYyzy2eh+OTIC
            13TYWDU4+3/LH17rc2HOJ8NbDq5EuWptzBJ6zw*D9AcEzQY+kBvuC4Z5AHJMz/r++xOcu
            g1hNj+5xIrcI8X7o/I+MYL9IJ6saxF6Aq24zwsc?2JI8FJujWfUChIG4qcr368uI*+d
            y7h1p7Io4vpF++9CY5o7+hgh
          </CipherValue>
        </CipherData>
      </EncryptedData>
    </soapenv:Body>
</soapenv:Envelope>
```

- The header includes the mustUnderstand="1" attribute, which indicates that either this header must be processed or a SOAP fault thrown.
- The <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/> child element of the <EncryptedKey> element in the SOAP header specifies the algorithm used to encrypt the secret key.
- The <ds:KeyName> element contains the distinguished name that uniquely identifies the CICS certificate that has been used to encrypt the message. This is used by CICS to identify the CICSCERT X.509 certificate, the private key of which CICS uses to decrypt the message.
- The <CipherValue> element in the SOAP header contains the encrypted secret key that was used to encrypt the message.
- The <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/> child element of the <EncryptedData> element in the SOAP body specifies the algorithm used to encrypt the message.
- The <CipherValue> element in the SOAP body contains the encrypted message.

The corresponding encrypted response message sent by CICS is shown in Example 9-26.

Example 9-26 Encrypted SOAP response message

```
<?xml version="1.0" encoding="UTF8" standalone="no" ?>
  <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <SOAP-ENV:Header>
      <wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis
        -200401-wss-wssecurity-secext-1.0.xsd" SOAP-ENV:mustUnderstand="1"
        xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssp
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000402

```
curity-utility-1.0.xsd"
xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
<wsse: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="x509certOO">MIIJ2QIJAOCAaYqOwIBAgIEN6OCzANBgkqhkiG9wOBAQUFADBN
    MQswCQYDVQQGEwJFUzELMAkGA1UECBMCQVAxCzAJBgNVBAcTAkFMMQswCQYDVQQK
    EwJFUzEPMA0GA1UECxMGQVBBTkVUMQswCQYDVQQDEwJFUzAeFw0wNjA5MjcxMTIx
    MjJaFw0wNjEwMjcxMTIxMjJaMEwxCzAJBgNVBAYTAkVTMQswCQYDVQQIEwJBUDEL
    MAkGA1UEBxMCQUwxDzANBgNVBAoTBkFQQU5FVDEPMA0GA1UECxMGQVBBTkVUMQsw
    CQYDVQQDEwJFUzCCAbcwggEsBgcqhkjOOAQBMIIBHwKBgQD9f1OBHXUSKVLfSpwu
    7OhpJ2PX0Bmi7+wH3RVDjzfZj6DbKxESZlZn2fnXWcihuMNhXJGcht2krL8vqLa1
    cJMCVKklcEVMd/JT4gUkvo7Tgo0yTPBVuPksa2ZnsV3EMYqQJennP1IKAvTrQwSR
    AjgYmLc1nOqMNFXRqi7Ox4TWLLIbFwSNDFobZ9tv1Cv1CjgcJ1in5W1DCZoq1AAo
    gYEAkOIvvGnrAi8XqGZm8FoZ1Xfo6PBVvNVIXwSSL1pvrokJ1KSkRZlMJeec3pvX
    mOuYmBQXx+6qNi5RpOyM+8FqQHnoLGT2Ranr8cjWggo1qxFoISvlBSjXvT3BZVWq
    jRoLBaB6cAgZNY2DCqdKeLLT3+5xlv1t1A6sCIDAQABoxowGDAJBgNVHRMEAjAA
    MAsGA1UdDwQEAwIE8DANBgkqhkiG9wOBAQUFAAOBgQBPTSHMA4vdoHIyexvi1H5p
    Q3n+GGdMJWAXP/DcMO+fgYaYr1l7mMBMCJDirLGrXYqjzyq2/yPAnAYFAkr+YJMO
    M7ykztCyJGchY97w75Nj/gyVBAAQjFTATMBEGA1UdDgQKBAhJYPBmS47GSTAVBgkq
    hkiG9w0BAQUFAAOBgQBPTSHMA4vdoHIPIbI8gNnUSwlsBm+xBPJiJqiXOrvYH4
    HSfpH+k4bu4fNDjO7YGSCrqSpnqGVYuHKDcPLb/c6YuDGkxr1AwCGR9+E1EJ1b
    U+MB66HqGN/qAT+glZuYrPkm2Cc0FURIQYcMMtn7+JBIc6oUp+GPQ==
</wsse:BinarySecurityToken>
<xenc:EncryptedKey xmlns:xenc="http://www.w3.org/2001/04/xmlenc#">
  <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
  <ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#x509cert00"
        ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-
        wss-x509-token-profile-1.0#X509"/>
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
  <xenc:CipherData>
    <xenc:CipherValue>SLqnQ7hId/2CugP94jAGE3M5xEJDBZMBVsrF8SkW/g==Cad2
      roLHaNVcLAq==g==Cn7ihP5JifGnC6bbqxsOxJA5+SuKC4IP5zPdK4c78uEG3Eeyrc
      ZRCmYkm3MOcECD9v2xwn9J/OgfsUuXEL3+9CwvcLr49KisCC8BndvCH3=
    </xenc:CipherValue>
  </xenc:CipherData>
  <xenc:ReferenceList>
    <xenc:DataReference URI="#Enc1"/>
  </xenc:ReferenceList>
</xenc:EncryptedKey>
</wsse:Security>
</SOAP-ENV:Header>
<SOAP-ENV:Body>
  <xenc:EncryptedData xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
    Id="Enc1" Type="http://www.w3.org/2001/04/xmlenc#Content">
    <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
    <xenc:CipherData>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000403



- The <wsse:BinarySecurityToken> contains the base64binary encoding of the WebSphere certificate. This includes the public key that WebSphere uses to decrypt the secret key.
- <wsse:SecurityTokenReference> is a child element of the <ds:KeyInfo> element which contains a reference to the BinarySecurityToken.

The rest of the SOAP headers in the response message are similar to those in the request message.

SOAP fault messages

In this section we show some of the SOAP faults that you may see when testing encryption processing.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000404

- Figure 9-48 shows the browser response when CICS is expecting an encrypted message but the service requester does not encrypt the request.

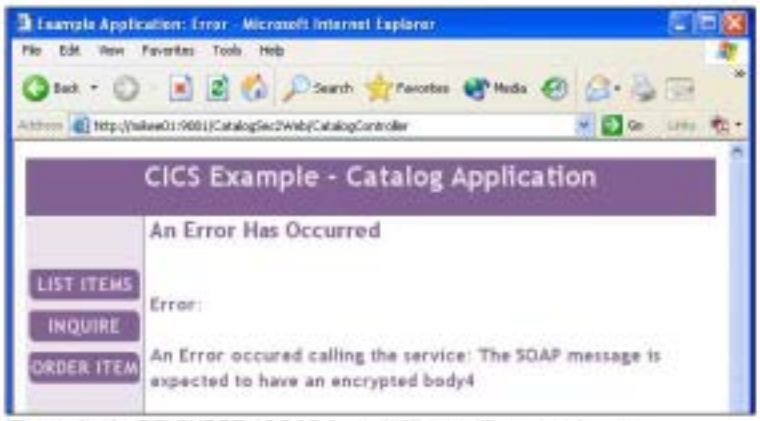


Figure 9-48 DFHWSSE1 SOAP fault: X.509 certificate not known

The SOAP fault for this message, shown in Example 9-27, highlights that CICS is expecting an encrypted message.

Example 9-27 DFHWSSE1 SOAP fault - X.509 certificate not known

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>wsse:FailedCheck</faultcode>
      <faultstring>INVALID_SOAP_REQUEST</faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
          <message>The SOAP message is expected to have an encrypted body
          </message>
          <errorcode>4</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 405/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000405

- Figure 9-49 shows the browser response when CICS is attempting to decrypt an encrypted message but ICSF is not available.

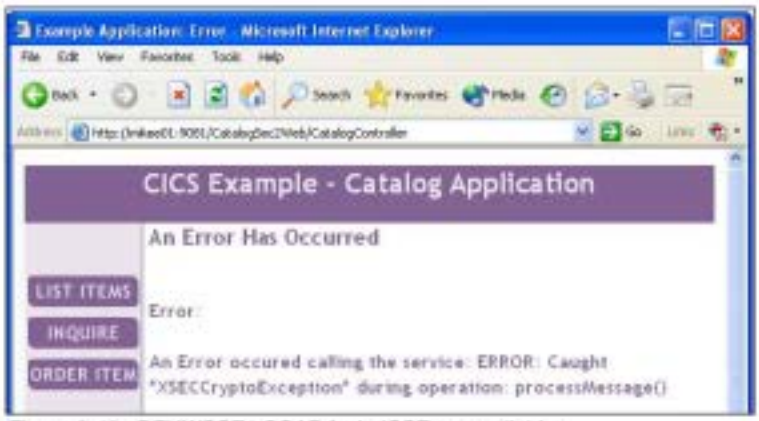


Figure 9-49 DFHWSSE1 SOAP fault: ICSF not available

The SOAP fault for this message, shown in Example 9-28, shows that there is an error creating RSA key token.

Example 9-28 DFHWSSE1 SOAP fault - ICSF not available

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>wsse:FailedCheck</faultcode>
      <faultstring>ERROR: Caught "XSECCryptoException" during operation:
        processMessage()</faultstring>
      <detail>
        <e:myfaultdetails xmlns:e="http://www.ibm.com/software/htp/cics/wssec">
          <message>ZosCryptoKeyRSA::privateDecrypt - Error encoding under RSA
            key</message>
          <errorcode>1</errorcode>
        </e:myfaultdetails>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 406/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000406

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000407

Part 4

Transaction management

In this part, we begin by providing an introduction to Web services atomic transactions. We then outline the steps for enabling WS-Atomic Transaction (WS-AT) support in CICS. Finally, we show several scenarios that demonstrate how you can synchronize WebSphere and CICS updates using the WS-AT standard.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000408

Document Id: [illegible]
Client Security: [illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000409

10

Introduction to Web services: Atomic transactions

We begin this chapter by describing an example of a classic transaction. We expect that most of our readers will be familiar with examples similar to ours. Then we map our classical transaction to a Web services *atomic transaction* as a means of introducing the new terminology that we use in this and succeeding chapters: *coordinator*, *transactional context*, *activation service*, *registration service*, and *completion protocol*. We continue by providing an overview of the flows and message contents prescribed by the three specifications upon which CICS TS V3.1 support for atomic transactions is built:

- Web Services - Addressing (WS-Addressing or WS-A)
- Web Services - Coordination (WS-Coordination or WS-C)
- Web Services - Atomic Transaction (WS-Atomic Transaction or WS-AT)

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000410

10.1 Beginner's guide to atomic transactions

We begin by describing an example of a classic transaction. For this discussion we borrow freely from the paper *Four Web Services Atomic Transaction operations: Beginner's guide to classic transactions, data recovery, and mapping to WS-Atomic Transactions* which Thomas Freund and Daniel House published on September 2, 2004, on the IBM developerWorks Web site at:

http://www-128.ibm.com/developerworks/webservices/library/ws-atmtrans1/

Not losing money is quite important. Just ask Waldo. Waldo's situation typifies the need for a transaction. Waldo uses a Web browser or an Automatic Teller Machine (ATM) to move some money from one account to another account. These accounts may be in different branches of the same financial institution, or they may be in different institutions.

It is never acceptable to Waldo for his money to disappear. Should Waldo ever doubt the safety of his money, he would probably switch financial institutions.

Waldo's money is represented by data in two databases that cooperate to ensure that the data they contain is always in a known and consistent state. That is, these two databases allow actions or tasks between them to be within a common activity or work scope as shown in Figure 10-1. Put yet another way, a single transaction can manipulate data in both databases and *something* will guarantee that only one of two possible outcomes occurs: *all* the changes are successfully made or *none* of them is made at all.

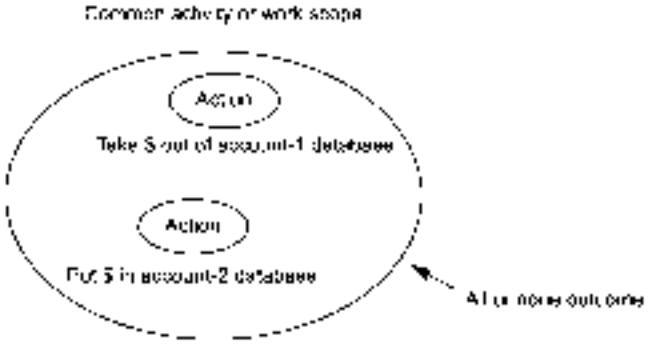


Figure 10-1 Common activity encompasses various recoverable actions

The *something* that guarantees the common outcome of all the actions is a protocol supported by both databases, and some supporting middleware. The

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000411

protocol the databases use to keep data (such as Waldo's balances) coordinated is called *two phase commit*, or simply 2PC. Our example uses a common variation of 2PC called *presumed abort*, where the default behavior in the absence of a successful outcome is to roll back or undo all the actions in the activity.

From a programming perspective, there are different ways to specify that multiple actions should be within the scope of a single transaction. One particularly clear way to specify transactional behavior is shown in Example 10-1. The code is the small piece of logic running somewhere behind the ATM Waldo is using—perhaps in the data center of one of the financial institutions involved.

Example 10-1 Pseudo-code for Waldo's transaction

```
TransferCash(fromAcct, toAcct, amount)
    BeginTransaction
    fromAcct = fromAcct - amount
    toAcct   = toAcct + amount
    CommitTransaction
Return
```

10.1.1 What is a classic transaction

For our simple purposes, a *recoverable* action is anything that modifies protected data. For example, taking money out of one of Waldo's accounts (fromAcct – fromAcct - amount) is a recoverable action that can be reversed up to the end of the transaction. A *classic* transaction, then, is just a grouping of recoverable actions, the guaranteed outcome of which is that either all the actions are taken, or none of them is taken (see Figure 10-1).

In Waldo's case, his transaction is composed of two actions, taking money out of one account and putting money into another account. It's okay for both of these actions to occur, and it's even okay if neither of these actions occurs. It's never okay for one action to occur without the other also occurring, which would result in corrupt data and either Waldo's net worth or the bank's assets disappearing or appearing from nowhere. Hence, both actions need to be within a single transaction with a single outcome: either both actions occur (a *commit* outcome), or neither action occurs (a *rollback* outcome).

Assuming no errors happen, the code in Example 10-1 shows that a commit outcome is desired. The code could just as easily have specified rollback instead of commit (for when Waldo presses the Cancel key on the ATM), which means reverse all actions in the transactional work scope (between beginning and end). The transaction monitor, which is the underlying middleware helping the code in Example 10-1 support transaction processing, would automatically specify

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000412

rollback if the program suffered an unhandled exception. Such an automatic rollback on the part of the transaction monitor is a protection mechanism to make sure that data is not corrupted. For example, even if the ATM application fails unexpectedly, the middleware will "clean up" and guarantee the outcome.

Now let's see how one common variant of 2PC, *presumed abort*, can be used to effect Waldo's transaction and move money from one account to another in a recoverable way. A key part of this illustration is to see that no matter what kind of failure occurs, data integrity is preserved and Waldo remains a loyal customer.

Figure 10-2 shows Waldo's transaction on a time line with all of the interacting components needed to execute the logic shown in Example 10-1.

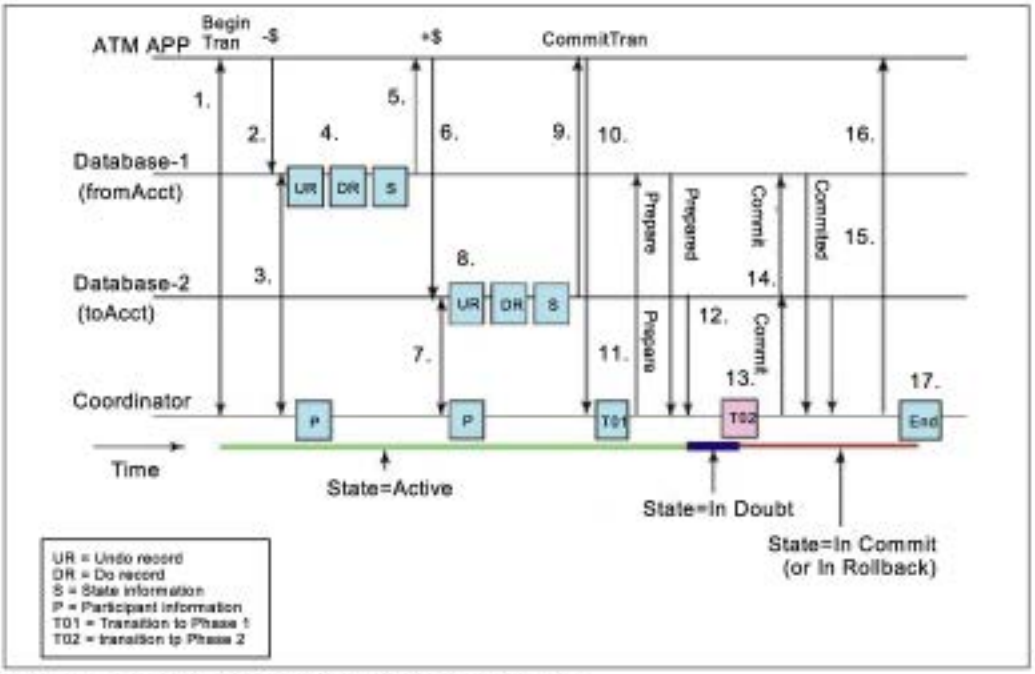


Figure 10-2 Behind the scenes of Waldo's ATM transaction

The top line represents the ATM application itself. The next two lines represent the account databases that the application manipulates. The databases will be transactional participants. The next line is a transactional coordinator, or middleware, which will orchestrate the 2PC protocol. The line at the very bottom indicates the state of Waldo's transaction at different points in time. The state of the transaction dictates recovery processing in the event of a failure.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 413/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000413

The lines for Database-1, Database-2, and Coordinator represent both time (flowing left to right) and also some key records recorded onto a recovery log. These records include images of the data before it is modified (Undo records), images of the data after it has been modified (Do records), and state information. The recovery log is used to ensure data integrity during recovery processing.

Now let's walk through Waldo's transaction. In the following discussion, when we talk about *the ATM application* take it to mean either the application itself, or some middleware supporting the application. For example, when we say the application begins a transactional scope, it could be that middleware begins the transactional scope on behalf of the application.

Here is narration to explain the numbered steps shown in Figure 10-2:

1. The ATM application indicates the beginning of a transactional scope. The Coordinator creates a context for this transaction; the context includes a unique identifier and some other information about the transaction. Importantly, this transaction context flows back to the application. The context flows with other interactions between the application and resource managers. It is the context that helps glue together a whole set of actions into one transactional activity.
2. The application takes money out of Database-1. The context (from step 1) is inserted into this flow.
3. Database-1 sees the request for action, but also sees the transactional context. Database-1 uses this context to contact the transactional Coordinator and register interest in this transaction or activity (so that the Coordinator will help Database-1 through 2PC processing later to guarantee a commit or rollback outcome of all actions). The Coordinator remembers that Database-1 is a participant in the transaction.
4. Database-1 looks at the request to modify recoverable data. It writes records to a recovery log, plus transaction state information. One record describes the database change to be made if the decision later is to commit (the Do record). The other record describes the database change to be made if the decision is to rollback (the Undo record).
   - In this case, the Undo record says *make Waldo's balance = X* and the Do record says *make Waldo's balance = X - $* (X is the amount of the balance before this transaction ever started and $ is the amount to transfer). Notice that we are only looking at the recovery log – not database files.
   - The Do records are not strictly required if Database-1 makes database file updates when the application requests it to. Instead of waiting, however, waiting to write the data can have advantages for performance and concurrency. In addition, the Do records may be used for audit or other advanced reasons. Since they are so useful, our example databases use them.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000414

5 Return to the application.

6 Similarly to step 2, the application makes a request to manipulate the other database, Database-2. The application wants to add in the amount taken out of Database-1.

7 Database-2 registers interest in the transaction with the Coordinator the same way Database-1 did. The Coordinator remembers that Database-2 is a participant in the transaction.

8 Database-2 writes Undo and Do records and state information to its recovery log, again just as Database-1 did.

9 Return to the application.

10. The application chooses to commit the transaction. The Coordinator now takes over. When **Commit** is received, the Coordinator writes a log record indicating that Phase 1 of 2PC has begun.

11. In Phase 1, the Coordinator goes down the list of all participants (Database-1 and Database-2 in this example) who expressed interest in this transaction asking each one to **Prepare**. Prepare means get ready to receive the order to either commit or rollback.

12 Database-1 and Database-2 both respond with **Prepared**, meaning that they are ready to be told the final outcome (commit or rollback all the changes made) and support it.

   - They must have committed something (at least on their logs) by this point, because responding **Prepared** means they guarantee being able to commit or rollback when told - actions up to this point were just tentative.
   - If either Database had some kind of failure preparing, it would respond **Aborted** instead of **Prepared**, and the Coordinator would broadcast **Rollback** to all participants.

13 The Coordinator forces a log record indicating a Transition to Phase 2 (TD2).

   Once this record is hardened on a log, we know and have recorded that:

   - All participants are prepared to go either way (commit or rollback).
   - The ultimate outcome of the transaction is known (commit in our example).
   - The outcome is guaranteed by recovery processing.

   If this record fails to make it to the log for any reason, the ultimate outcome will be to roll back (we are using *presumed abort* in this example). The recovery processing will enforce the outcome.

14 The Coordinator informs each participant that the decision is to commit the changes. The participants can then do whatever they need to do, such as perhaps writing the results to the real database data.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000415

15. The participants return to the Coordinator with **Committed**. Once the Coordinator knows that all the participants acknowledged the **Commit** order with **Committed**, it can forget about this transaction because the transaction was acknowledged by all to be done.

16. Return to the application.

17. At some point, since it knows the participants have succeeded in the 2PC flow by acknowledging the common outcome, the Coordinator writes an *End* indicator on its log.

10.1.2 Mapping from classic transactions to WS-Atomic Transaction

In Figure 10-2 on page 384 we did not mention how Database-1 contacted the Coordinator, nor did we specify how the application called the databases. In fact we didn't specify the mechanisms for anything to contact anything else. In the past these were mostly non-universal mechanisms that sometimes only worked between certain combinations of entities (applications, resource managers, and coordinators or transaction monitors).

The combination of Web services, WS Coordination (WS-C) and WS Atomic Transaction (WS-AT) maps all of the flows shown in Figure 10-2 on page 384 and specifies precise communications mechanisms for achieving the same results. However, instead of only working between certain combinations, the Web services based flows can work with just about anything.

Figure 10-3 illustrates how the classic flows are converted to Web services. Significantly changed steps are described following the figure. As before, when we say *application*, take it to mean the application or some helper middleware. Likewise, when we say *database*, it might mean the actual database, or some helper middleware.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000416

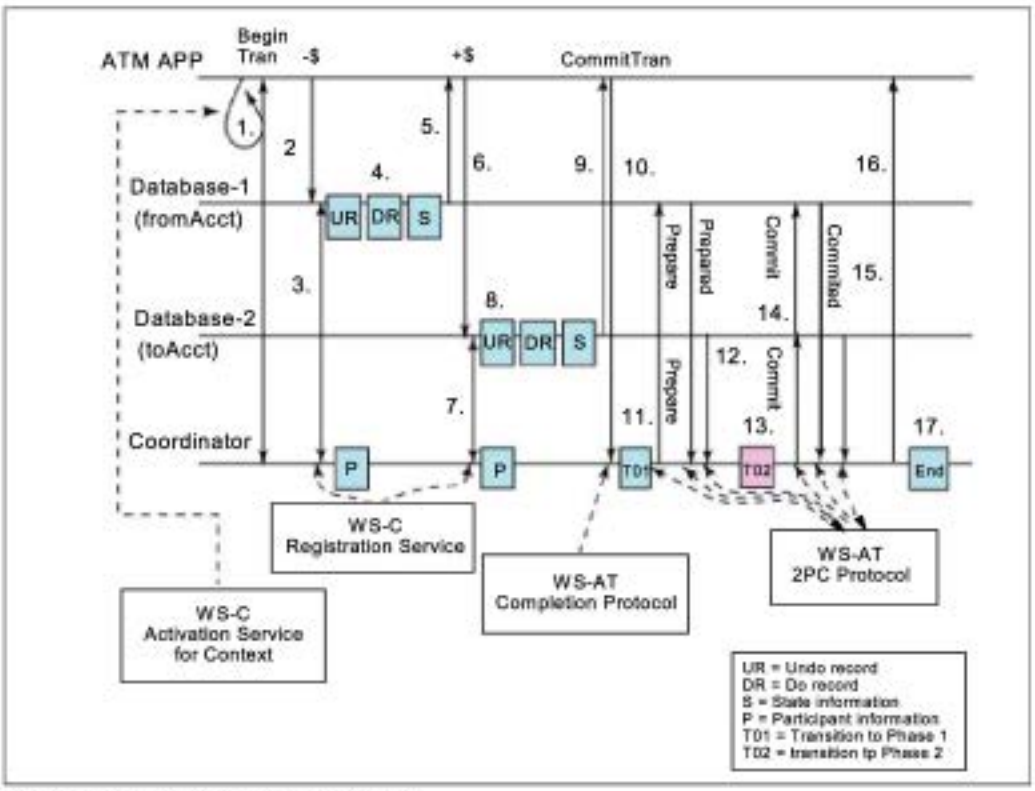


Figure 10-3 Waldo's transaction revisited

1. The application uses the Activation Service defined in WS-C to obtain a transactional context.
2. The application invokes a Web service exposed by Database-1 (alternatively, exposed by an application server that then talks to Database-1) to subtract money from Waldo's balance. The context flows along with the Web service invocation, although the application is not aware of that.
3. Database-1 uses information in the context to invoke the Registration Service defined in WS-C to register interest in this transaction.
4. No change.
5. No change.
6. The application invokes a Web service exposed by Database-2 to add money to Waldo's balance. Just like in step 2, the context flows along with the Web service invocation.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 417/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000417

7 Just like step 3, Database-2 uses information in the context to invoke the Registration Service and register interest in this transaction.

8 No change.

9 No change.

10. The application uses the Completion Protocol defined in WS-AT to indicate that it wishes to commit the transaction.

11 to 15. Databases and the Coordinator participate in 2PC flows as defined in the WS-AT 2PC Protocol.

From Figure 10-3 it is clear that atomic transactions using Web services (WS-C and WS-AT) are substantially the same as without Web services (Figure 10-2 on page 384, for example). The primary differences are almost entirely from the outside and involve *how entities communicate with each other, not the substance of what they communicate*. However, these differences in how the entities communicate have a big impact on flexibility and interoperability.

You can achieve universal interoperability with Web services because instead of changing resource manager X to interoperate with transaction monitor Y, you can change both X and Y to use Web services and then interoperate with many other resource managers and transaction monitors. So instead of two-at-a-time interoperability, or interoperability only within a specific kind of domain, n-way universal interoperability is possible.

Recovery processing using Web services between the interested parties is the same as before Web services. Resource managers are the only ones who know their resources and how to commit them or roll them back.

As an example, suppose that Database-1 fails between steps 5 and 6 in Figure 10-3. Database-1 comes back up and, just like before Web services, it reads its log, notices that it has an incomplete transaction, and realizes that it needs to contact the Coordinator. Information on how to contact the Coordinator is in the state saved on its recovery log. With Web services it will be an end point reference (as defined in WS-Addressing; see "Endpoint references" on page 391). Database-1 contacts the Coordinator at that end point reference with a message defined in WS-AT called **Replay**. Replay causes the Coordinator to resend the last protocol message to Database-1, which lets Database-1 deduce the transaction state and then apply the appropriate recovery rule. In our example the Coordinator tells Database-1 that it has no knowledge of this transaction. Database-1 therefore applies its Undo record, making the data consistent again.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000418

Important: WS-AT is a two-phase commit transaction protocol that is suitable for short duration transactions only. WS-AT is well suited for distributed transactions within a single enterprise, but is it is generally not recommended that WS-AT transactions be distributed across enterprise domains. Inter-enterprise transactions typically require a looser semantic than two-phase commit.

10.2 WS-Addressing

Figure 10-3 on page 388 shows several messages flowing:

- The application sends a message to the Activation Service asking for a transactional context.
- The Activation Service sends a response containing a transactional context to the application.
- Database-1 and Database-2 each send a **`Register`** message to the Registration Service and receives a reply.
- The application sends a **`Commit`** message to the Coordinator.
- The Coordinator sends **`Prepare`** and **`Commit`** messages to Database-1 and Database-2.
- Database-1 and Database-2 each send **`Prepared`** and **`Committed`** messages to the Coordinator.

The application, Database-1, Database-2, the Activation Service, the Registration Service, and the Coordinator are endpoints for these messages. As with messages in the everyday business world, we need a way to identify the recipient of each message, the sender of the message, what previous message (if any) the message relates to, what action we want the recipient to take, and where the recipient should send the reply (if any) to the message. Furthermore, we want to be able to do this in a way that does not depend on the transport mechanism (such as HTTP or WebSphere MQ) that we use to send the message.

To this end IBM, Microsoft, Sun™ Microsystems™, BEA, and SAP® formally submitted the *Web Services- Addressing (WS - Addressing)* specification to the World Wide Web Consortium (W3C) on 10 August 2004. You can find this specification at:

`http://www.w3.org/Submission/ws-addressing`

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 419/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000419

Note: As the specification has moved through the W3C standards process, it has been divided into three parts:

- Web Services Addressing 1.0 - Core

  This is currently a W3C Candidate Recommendation dated 17 August 2005.

- Web Services Addressing 1.0 - SOAP Binding

  This is also currently a W3C Candidate Recommendation dated 17 August 2005.

- Web Services Addressing 1.0 - WSDL Binding

  This is currently a W3C Working Draft dated 13 April 2005.

However, the WS-Coordination and WS-Atomic Transaction specifications that we discuss later in this chapter are based on the 10 August 2004 specification, and, therefore, so is CICS TS V3.1 support for Web services.

All information items defined by the 10 August 2004 specification are identified by the XML namespace URI:

```
http://schemas.xmlsoap.org/ws/2004/08/addressing
```

We associate the namespace prefix *wsa* with this namespace by using the attribute

```
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
```

The specification defines two constructs:

- Endpoint references
- Message information headers

10.2.1 Endpoint references

A Web service *endpoint* is a (referenceable) entity, processor, or resource to which Web services messages can be addressed.

An *endpoint reference* conveys the information needed to address a Web service endpoint. As we shall see later:

- An endpoint reference for the Registration Service forms part of the coordination context.
- An endpoint reference for the participant's Protocol Service forms part of the Register request.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 420/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000420

- An endpoint reference for the coordinator's Protocol Service forms part of the response to a Register request.

Example 10-2 shows the pseudo schema for an EndpointReference element.

Example 10-2 Pseudo schema for EndpointReference element

```
<wsa:EndpointReference>
   <wsa:Address>.....</wsa:Address>
   <wsa:ReferenceProperties>.....</wsa:ReferenceProperties>
   <wsa:ReferenceParameters>.....</wsa:ReferenceParameters>
   <wsa:PortType>.....</wsa:PortType>
   <wsa:ServiceName PortName="...">.....</wsa:ServiceName>
   <wsp:Policy>.....</wsp:Policy>
</wsa:EndpointReference>
```

For each child element of EndpointReference, Table 10-1 describes what the element contains, the minimum number of times the element can be used, and the maximum number of times the element can be used.

Table 10-1 Children of the EndpointReference element

Element	Description	Min	Max
Address	Contains an address URI that identifies the endpoint. This may be a network address or a logical address.	1	1
ReferenceProperties	Contains child elements each of which represents an individual reference property. The number of child elements is not limited	0	1
ReferenceParameters	Contains child elements each of which represents an individual reference parameter. The number of child elements is not limited	0	1
PortType	Contains the name of the primary portType of the endpoint being conveyed	0	1
ServiceName	Contains the name of the WSDL <service> element that contains a WSDL description of the endpoint being referenced. The service name provides a link to a full description of the service endpoint. The ServiceName element may optionally have a PortName attribute which specifies the name of the specific WSDL <port> definition in that service which corresponds to the endpoint being referenced.	0	1
Policy	Contains an XML policy element as described in WS-Policy that describes the behavior, requirements, and capabilities of the endpoint.	0	No limit

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000421

Here is the difference between a reference property and a reference parameter:

- A *reference property* is required to identify the endpoint. It is required to properly dispatch a message to an endpoint at the endpoint side of the interaction.
- A *reference parameter* is required to facilitate a particular interaction with the endpoint. It is required to properly interact with the endpoint.

Tip: In "Transaction scenarios" on page 435 we look at the messages that flow between CICS TS V3.1 and WebSphere Application Server V6.0. In these messages we see only the following children of the `EndpointReference` element:

- Address
- ReferenceProperties

Example 10-3 shows an endpoint reference for the Registration Service running in a CICS TS V3.1 region that is monitoring port 15301 on a z/OS system whose IP address is MVSG3.mop.ibm.com. The endpoint reference has two reference properties: `UOWID` and `PublicID`. We replaced the ending characters of the `PublicID` property with ... for brevity.

Example 10-3 Sample EndpointReference element for Registration Service in CICS

```
<wsa:EndpointReference
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
      xmlns:cicswsat="http://www.ibm.com/xmlns/prod/CICS/pipeline">
   <wsa:Address>
       http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
   </wsa:Address>
   <wsa:ReferenceProperties>
      <cicswsat:UOWID>BDFC52CD7C57D466</cicswsat:UOWID>
      <cicswsat:PublicId>310FD7E2E2...</cicswsat:PublicId>
   </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

Example 10-4 shows an endpoint reference for the Registration Coordinator Port running in a WebSphere Application Server V6.0 region that is monitoring port 9080 on a Windows system whose IP address is 9.100.199.156. The endpoint reference has two reference properties: `txID` and `instanceID`. We replaced the ending characters of these properties with ... for brevity.

Example 10-4 Sample EndpointReference element for Registration Service in WAS

```
<wsa:EndpointReference
   xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
   xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension">
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net



Page 422/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000422

```
    <wsa:Address>
       http://9.100.199.138:9080/_IBMSYSAPP/wscoor/services/Registration
       CoordinatorPort
    </wsa:Address>
    <wsa:ReferenceProperties>
       <websphere-wsat:txID>
            com.ibm.ws.wstx:0000010d:d70ad76c0000000c0000000bcbbb1a/db7e...
       </websphere-wsat:txID>
       <websphere-wsat:instanceID>
            com.ibm.ws.wstx:0000010d:d70ad26c0000000c0000000bcbbb1a/db7e...
       </websphere-wsat:instanceID>
    </wsa:ReferenceProperties>
</wsa:EndpointReference>
```

10.2.2 Message information headers

Example 10-5 shows pseudo schema for the message information headers defined in the WS-Addressing specification.

Example 10-5 Pseudo schema for message information header elements

```
<wsa:MessageID>.....  .....</wsa:MessageID>
<wsa:RelatesTo RelationshipType="...">...</wsa:RelatesTo>
<wsa:To>.....</wsa:To>
<wsa:Action>.....</wsa:Action>
<wsa:From>.....</wsa:From>
<wsa:ReplyTo>.....</wsa:ReplyTo>
<wsa:FaultTo>.....</wsa:FaultTo>
```

Table 10-2 describes what each message information header element contains, the minimum number of times the element may be used, and the maximum number of times the element may be used.

Table 10-2 Message information header elements

Element	Description	Min	Max
MessageID	Contains a URI that uniquely identifies this message in space and time.	0 (but see Note 1)	1
RelatesTo	Contains a URI that corresponds to a related message's MessageID property. The RelatesTo element has an optional RelationshipType attribute that indicate the type of relationship this message has to the related message. The specification defines one relationship type, namely wsa:Reply When absent, the implied value of this attribute is wsa:Reply.	0	No limit

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000423

Element	Description	Min	Max
To	Contains a URI that specifies the address of the intended receiver of this message.	1	1
Action	Contains a URI that uniquely identifies the semantics implied by this message	1	1
From	Contains an endpoint reference that identifies the endpoint from which the message originated.	0	1
ReplyTo	Contains an endpoint reference that identifies the intended receiver for replies to this message.	0 (but see Note 2)	1
FaultTo	Contains an endpoint reference that identifies the intended receiver for faults related to this message.	0 (see also Note 3)	1

Table notes:

1. If ReplyTo or FaultTo is present, MessageID must be present.
2. If a reply is expected, ReplyTo *must* be present
3. When formulating a fault message, the sender *must* use the contents of the FaultTo element of the message to which the Fault reply is being sent. If the FaultTo element is absent, the sender *may* use the contents of the ReplyTo element to formulate the fault message. If both the FaultTo element and the ReplyTo element are absent, the sender *may* use the contents of the From element to formulate the fault message. The FaultTo element may be absent if the sender cannot receive fault messages.

Example 10-6 shows an example of a set of message information headers

Example 10-6 Sample message information header elements

```
<wsa:To>
   http://9.100.199.156:9080/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
</wsa:To>
<wsa:Action>
   http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register
</wsa:Action>
<wsa:MessageID>UUID-A6F0F3C1-0333477607977B5C</wsa:MessageID>
<wsa:ReplyTo>
   <wsa:Address>
      http://MYSQL.rop.ibm.com:1630/cics/wsat/RegistrationService
   </wsa:Address>
   <wsa:ReferenceProperties>
      <cicswsat:UOWID>0D7CE2C07157D466</cicswsat:UOWID>
      <cicswsat:PublicID>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000424

```
            3303D3D2D2...
        </wsaw:PublicID>
    </wsa:ReferenceProperties>
</wsa:ReplyTo>
```

The To header shows that the message is being sent to the Registration Coordinator Port running in a WebSphere Application Server V6.0 region that is monitoring port 9080 on a Windows system whose IP address is 9.100.199.156.

The Action header indicates that the sender wishes to register with the Registration Coordinator Port.

The ID of the message, `FTAI-MSG-A6P0T3C1-BB3342265277350`, uniquely identifies the message in space and time.

- `A6P0T3C1` is the VTAM APPLID of the CICS TS V3.1 region that sent the message.
- `BB3342265277350` is the abstime value returned by an EXEC CICS INQUIRE TIME issued in that CICS TS V3.1 region.

The ReplyTo header shows that the reply to this message should be sent to the Registration Service running in a CICS TS V3.1 region that is monitoring port 15301 on a z/OS system whose IP address is MVSG3.mop.ibm.com.

10.2.3 SOAP binding for endpoint references

When a SOAP message must be addressed to an endpoint, the information contained in the endpoint reference is mapped to the SOAP message by the following two rules:

- The contents of the Address element in the endpoint reference is copied to the To message information header of the SOAP message.
- Each reference property or reference parameter is added as a header block in the new message.

Example 10-7 shows how we use these rules to address a message to the CICS Registration Service endpoint whose endpoint reference is given in Example 10-3 on page 393.

Example 10-7 SOAP message addressed to CICS RegistrationService endpoint

```
<S:Envelope xmlns:S="..." xmlns:wsa="..." xmlns:cicswsa="...">
   <S:Header>
      ...
      <wsa:To>
          http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
```


Document Id: [illegible]
Confidentiality: [illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000425

```
        </wsa:To>
        <cicswsat:UOWLID>BDF0E2C0215D466</cicswsat:UOWLID>
        <cicswsat:PublicID>3109F07E2E2...</cicswsat:PublicID>
        ...
    </S:Header>
    <S:Body>
        ...
    </S:Body>
</S:Envelope>
```

10.3 WS-Coordination

In Section 10.1 "Beginner's guide to atomic transactions" on page 382 and Section 10.2. "WS-Addressing" on page 390 we see that the application, Database-1, Database-2, the Activation service, the Registration service, and the Coordinator are endpoints for messages. We also see that:

- The application sends a message to the Activation service asking for a transactional context
- The Activation service sends a response containing a transactional context to the application.
- Database-1 and Database-2 each sends a **Register** message to the Registration service and receives a reply.

We still need to define what these messages should contain.

For this purpose IBM, Microsoft, and BEA published the *Web Services-Coordination (WS - Coordination)* specification in September of 2003; they updated it in November of 2004. Arjuna Technologies Ltd., Hitachi Ltd., and IONA Technologies joined IBM, Microsoft, and BEA in publishing *Web Services-Coordination (WS-Coordination) Version 1.0* in August of 2005. You can find these at:

```
http://www-128.ibm.com/developerworks/library/specification/ws-tx
```

Although CICS TS V3.1 was developed and tested when the November, 2004 version was the current version, the nature of the differences between the November, 2004 version and the August, 2005 version is such that it may accurately be said that CICS TS V3.1 also supports the August, 2005 version.

All information items defined by the November, 2004 and August, 2005 versions are identified by the XML namespace URI:

```
http://schemas.xmlsoap.org/ws/2004/10/wscoor
```


Document Id: [illegible]
Confidential: [illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000426

We associate the namespace prefix *wscoor* with this namespace by using the attribute:

```
xmlns:wscoor="http://schemas.xmlsoap.org/ws/2004/10/wscoor"
```

The specification defines:

- A coordination service
- The following messages:
  - CreateCoordinationContext
  - CreateCoordinationContextResponse
  - Register
  - RegisterResponse

10.3.1 Coordination service

As shown in Figure 10-4, a *Coordination service* (or *Coordinator*) is an aggregation of the following services:

- Activation service

  When the application sends a `CreateCoordinationContext` element, the Activation service creates a new activity and returns its coordination context in a `CreateCoordinationContextResponse` element.

  The Coordination service may, but does not have to, support the Activation service.

  Note: Some products provide this as an external service, for others to call. CICS does not do this, and only supports the creation of coordination contexts internally, for use by the workloads that it manages.

- Registration service

  The Registration service defines a **`Register`** operation that allows a Web service to register to participate in a coordination protocol.

  The Coordination service must support the Registration service.

- A set of coordination Protocol services for each supported coordination type.

  These are defined in the specification that defines the coordination type (for example, in the WS-Atomic Transaction specification).

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000427

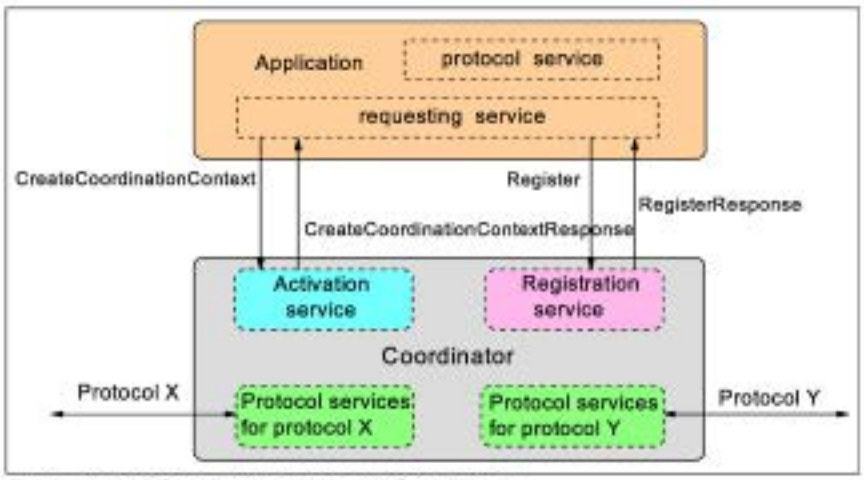


Figure 10-4 A Coordination service (or Coordinator)

10.3.2 CreateCoordinationContext

Example 10-8 shows the pseudo schema for the `CreateCoordinationContext` element.

Example 10-8 Pseudo schema for CreateCoordinationContext element

```
<wscoor:CreateCoordinationContext...>
   <wscoor:CoordinationType>.............</wscoor:CoordinationType>
   <wscoor:Expires>......................</wscoor:Expires>
   <wscoor:CurrentContext>..............</wscoor:CurrentContext>
</wscoor:CreateCoordinationContext>
```

For each child element of `CreateCoordinationContext`, Table 10-3 describes what the element contains, the minimum number of times the element can be used, and the maximum number of times the element can be used.

Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineHistory.net
Page 428/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000428

Table 10-3 Children of the CreateCoordinationContext element

Element	Description	Min	Max
CoordinationType	The unique identifier for the desired coordination type for the activity.	1	1
Expires	The expiration for the returned CoordinationContext expressed as an unsigned integer in milliseconds. Specifies the earliest point in time at which a transaction may be terminated solely due to its length of operation.	0	1
CurrentContext	The current Coordination Context.	0	1

Currently there are two specifications that define coordination types:

- WS-Atomic Transaction

  This specification defines the coordination type for atomic transactions where the results of operations are not made visible until the completion of the unit of work:

  `http://schemas.xmlsoap.org/ws/2004/10/wsat`

- WS-Business Activity

  This specification defines two coordination types for business activities:

  `http://schemas.xmlsoap.org/ws/2004/10/wsba/AtomicOutcome`

  `http://schemas.xmlsoap.org/ws/2004/10/wsba/MixedOutcome`

  Business activities have the following characteristics:

  - A business activity may consume many resources over a long duration.
  - There may be a significant number of atomic transactions involved.
  - Individual tasks within a business activity can be seen prior to the completion of the business activity, their results may have an impact outside of the computer system.
  - Responding to a request may take a very long time. Human approval, assembly, manufacturing, or delivery may have to take place before a response can be sent.
  - In the case where a business exception requires an activity to be logically undone, abort is typically not sufficient. Exception handling mechanisms may require business logic, for example in the form of a compensation task, to reverse the effects of a previously completed task.

  For example, selling a vacation is a business activity that may involve the travel agent in actions such as recording customer details, booking seats on

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000429

an aircraft, booking a hotel, booking a rental car, invoicing the customer, checking for receipt of payment, processing the payment, and arranging foreign currency.

Table 10-4 compares an atomic transaction with a business activity.

Table 10-4 Comparison of features of atomic transaction and business activity

Atomic transaction	Business activity
Short duration	Longer duration
Locks	Avoid locks
Suited for a more controlled environment	Suited for a loosely coupled environment
Classical resource manager mapping - think database (not business processes crossing business boundaries)	Business process mapping
Easier to think about and program ► Rollback or Commit ► Automatic rollback in case of abnormal termination	More complex ► Compensate
All resource managers move in one direction (everybody commits or rolls back in unison)	More flexible resource manager participation. They don't have to trust applications so much.

Note: CICS TS V3.1 does not support the WS-Business Activity specification. At this time there are no plans for future releases of CICS to support it either.

Example 10-9 shows an example of a `CreateCoordinationContext` element in which the coordination type is WS-Atomic Transaction.

Example 10-9 Sample CreateCoordinationContext element

```
<wscoor:CreateCoordinationContext>
   <wscoor:CoordinationType>
      http://schemas.xmlsoap.org/ws/2004/10/wsat
   </wscoor:CoordinationType>
   <wscoor:Expires>5000</wscoor:Expires>
</wscoor:CreateCoordinationContext>
```

10.3.3 CreateCoordinationContextResponse

The `CreateCoordinationContextResponse` element contains the `CoordinationContext` element.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 430/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000430

The CoordinationContext element contains four elements:

- A URI which identifies the CoordinationContext.
- The CoordinationType.
- The expiration period for the CoordinationContext.
- An endpoint reference for the Registration Service which is part of this Coordination service. Recall from "Endpoint references" on page 391 that this means that the CoordinationContext must contain the address of the Registration Service and may contain reference properties for the Registration Service.

Example 10-10 shows the pseudo schema for the CreateCoordinationContextResponse element.

Example 10-10 Pseudo schema for CreateCoordinationContext Response element

```
<wscoor:CreateCoordinationContextResponse>
   <wscoor:CoordinationContext>
      <wscoor:Identifier>.....................</wscoor:Identifier>
      <wscoor:CoordinationType>...............</wscoor:CoordinationType>
      <wscoor:Expires>........................</wscoor:Expires>
      <wscoor:RegistrationService>............</wscoor:RegistrationService>
   </wscoor:CoordinationContext>
</wscoor:CreateCoordinationContextResponse>
```

The application will place the CoordinationContext element within an application message to pass the coordination information to other parties. Conveying a CoordinationContext on an application message is commonly referred to as *flowing* the context. When a context is flowed as a SOAP header, the header must have the mustUnderstand attribute and the value of the mustUnderstand attribute must be true.

Note: When an application flows the context, it passes the address of its Registration Service and the coordination type.

Example 10-11 shows a CoordinationContext created by a CICS TS V3.1 region.

Example 10-11 Sample CoordinationContext created by CICS

```
<wscoor:CoordinationContext>
   <wscoor:Identifier>
      PIAT-CCON-A6P0T3C1-003322404576825C
   </wscoor:Identifier>
   <wscoor:CoordinationType
      http://schemas.xmlsoap.org/ws/2004/10/wsat
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000431

```
   </wscoor:CoordinationType>
   <wscoor:RegistrationService>
      <wsa:Address>
         http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
      </wsa:Address>
      <wsa:ReferenceProperties>
         <cicswsat:NetName>A6POT3C1</cicswsat:NetName>
         <cicswsat:Token>F0F0F0F0</cicswsat:Token>
         <cicswsat:UOWID>BCD88F2EB52B924C</cicswsat:UOWID>
      </wsa:ReferenceProperties>
   </wscoor:RegistrationService>
</wscoor:CoordinationContext>
```

Example 10-12 shows a `CoordinationContext` created by WebSphere Application Server V6.0.

Example 10-12 Sample CoordinationContext created by WebSphere

```
<wscoor:CoordinationContext>
   <wscoor:Expires>Never</wscoor:Expires>
   <wscoor:Identifier>
      com.ibm.ws.wstx:00000107d70ad2...
   </wscoor:Identifier>
   <wscoor:CoordinationType
      http://schemas.xmlsoap.org/ws/2004/10/wsat
   </wscoor:CoordinationType>
   <wscoor:RegistrationService>
      <wsa:Address>
         http://9.100.199.156:9080/_IBMSYSAPP/wscoor/services/RegistrationCoord
         inatorPort
      </wsa:Address>
      <wsa:ReferenceProperties>
         <websphere-wsat:txID xmlns:websphere-wsat="......">
            com.ibm.ws.wstx:00000107d70ad2...
         </websphere-wsat:txID>
         <websphere-wsat:instanceID xmlns:websphere-wsat="......"
            com.ibm.ws.wstx:00000107d70ad2...
         </websphere-wsat:instanceID>
      </wsa:ReferenceProperties>
   </wscoor:RegistrationService>
</wscoor:CoordinationContext>
```

Note: We noted from our tests that the content of the `Expires` element (that is, `Never`) is not an unsigned integer as required by the specification. This is planned to be changed in a future release of WebSphere Application Server.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 432/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000432

10.3.4 Register

Before we provide the details of the Register request, we consider some concepts in this section so that you don't lose sight of the 'big picture.'

The Coordinator provides the application with the Endpoint reference of its Registration service in the CreateCoordinationContextResponse. The application then knows where and how to send a Register request.

Figure 10-5 shows how Endpoint references are used during and after registration.

1. The Register message targets the Endpoint reference of the Coordinator's Registration Service and includes the Endpoint reference of the application's Protocol service as a parameter.
2. The Register Response includes the Endpoint reference of the Coordinator's Protocol service.
3. At this point, both sides have the Endpoint Reference of the other's Protocol service, so the protocol messages can target the other side.

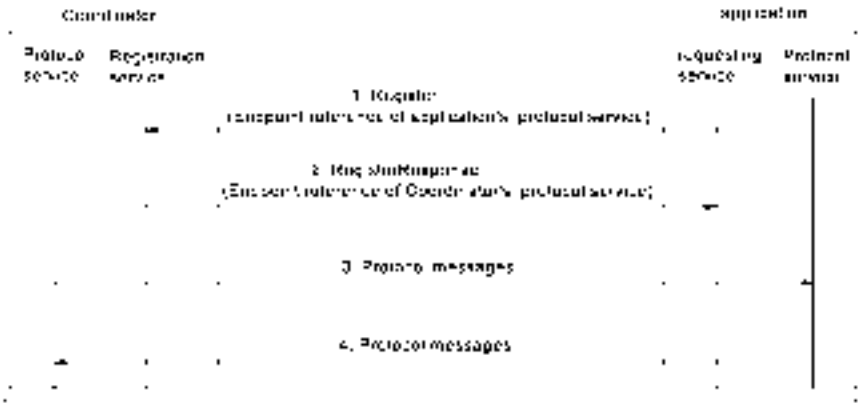


Figure 10-5 Using Endpoint references during and after registration

Now that you understand the big picture, we provide the details. Example 10-13 shows the pseudo schema for the Register element.

Example 10-15 Pseudo schema for the Register element

```
<wscoor:Register>
   <wscoor:ProtocolIdentifier>.....</wscoor:ProtocolIdentifier>
   <wscoor:ParticipantProtocolService>.....</wscoor:ParticipantProtocolService>
</wscoor:Register>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000433

The ProtocolIdentifier element contains a URI that provides the identifier of the coordination protocol selected for registration. The contents of the CoordinationType element of the CoordinationContext element determine the possible choices for the ProtocolIdentifier element as follows:

- If the CoordinationType is atomic transaction, then the ProtocolIdentifier must be one of the following:
  - http://schemas.xmlsoap.org/ws/2004/10/wsat/Completion
  - http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC
  - http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
- If the CoordinationType is business activity (either AtomicOutcome or MixedOutcome), then the ProtocolIdentifier must be one of the following:
  - http://schemas.xmlsoap.org/ws/2004/10/wsba/BusinessAgreementWithParticipantCompletion
  - http://schemas.xmlsoap.org/ws/2004/10/wsba/BusinessAgreementWithCoordinatorCompletion

The ParticipantProtocolService element contains the EndpointReference that the registering participant wants the Coordinator to use for the Protocol service.

Note: As we noted earlier, CICS TS V3.1 does not support the WS-Business Activity specification. Therefore, it does not support either the BusinessAgreementWithParticipantCompletion or the BusinessAgreementWithCoordinatorCompletion protocol identifiers.

Example 10-14 is an example of a Register element created by a CICS TS V3.1 region.

Example 10-14 Sample Register element

```
<wscoor:Register>
   <wscoor:ProtocolIdentifier>
      http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
   </wscoor:ProtocolIdentifier>
   <wscoor:ParticipantProtocolService>
      <wsa:Address>
         http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
      </wsa:Address>
      <wsa:ReferenceProperties>
         <cicswsat:UOWID>BDFC52CD7C57D466</cicswsat:UOWID>
         <cicswsat:PublicId>310FD7E2E2...</cicswsat:PublicId>
      </wsa:ReferenceProperties>
   </wscoor:ParticipantProtocolService>
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 434/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000434

```
</wscoor:Register>
```

Note: CICS TS V3.1 uses a single endpoint address for its Registration service, Protocol service, and fault messages. Other products, such as WebSphere Application Server V6.0, use separate addresses for each of these.

10.3.5 Register response

Example 10-15 shows the pseudo schema for the `RegisterResponse` element.

Example 10-15 Pseudo schema for the RegisterResponse element

```
<wscoor:RegisterResponse>
   <wscoor:CoordinatorProtocolService>.....</wscoor:CoordinatorProtocolService>
</wscoor:RegisterResponse>
```

The `CoordinatorProtocolService` element contains the `EndpointReference` that the Coordination service wants the registered participant to use for the Protocol service.

We note, once again, that when the application has received the `RegisterResponse`, each side has the `EndpointReference` of the other's Protocol Service.

10.3.6 Two applications with their own coordinators

In this section we see how two application services (App1 and App2) with their own coordinators (Coordinator A and Coordinator B) interact as an activity propagates between them. Coordinator A provides Activation service ASa, Registration service RSa, and Protocol service Pa. Coordinator B provides Activation service ASb, Registration service RSb, and Protocol service Pb. Figure 10-6 shows the two applications with their own coordinators.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 435/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000435

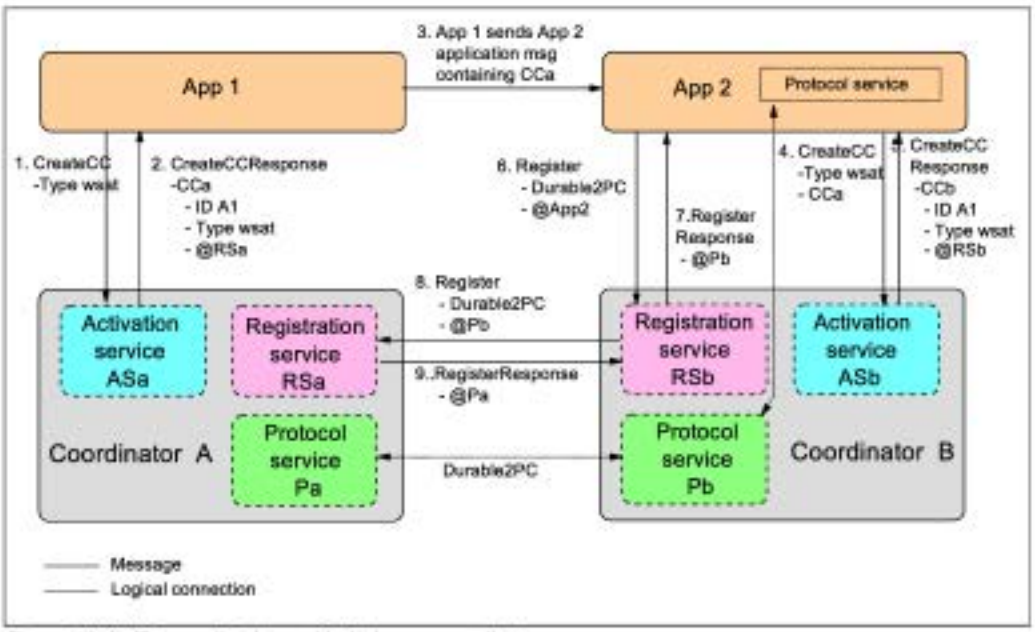


Figure 10-6 Two applications with their own coordinators

1. App1 sends to ASa a `CreateCoordinationContext` element that specifies a `CoordinationType` of wsat.
2. ASa sends to App1 a `CreateCoordinationContextResponse` element that includes `CoordinationContext` CCa. CCa contains an `Identifier` element whose content is A1, a `CoordinationType` element whose content is wsat, and a `RegistrationService` element that contains an `EndpointReference` to Coordinator A's Registration service RSa.
3. App1 then sends to App2 a SOAP message that has a SOAP header that contains CCa and that has a `mustUnderstand` attribute with a value of `True`.
4. App2 prefers Coordinator B, so it sends to ASb a `CreateCoordinationContext` element that specifies a `CurrentContext` of CCa.
5. Coordinator B creates its own `CoordinationContext` element CCb that contains the same `Identifier` and `CoordinationType` as CCa but with an `EndpointReference` to its own RegistrationService RSb.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 436/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000436

The *WS-Atomic Transaction* specification which we discuss in detail in "WS-Atomic Transaction" on page 408 states that:

If the CreateCoordinationContext request includes the CurrentContext element, then the target coordinator is interposed as a subordinate to the coordinator stipulated inside the CurrentContext element.

- If the CreateCoordinationContext request does not include a CurrentContext element, then the target coordinator creates a new transaction and acts as the root coordinator.

6. App2 determines the coordination protocols supported by the wsat coordination type and then sends a Register element to RSb. This Register element contains a ProtocolIdentifier of Durable2PC and an EndpointReference to App2's Protocol service.
7. RSb sends back to App2 a RegisterResponse element that contains an EndpointReference to Protocol service Pb. This forms a logical connection (which the Durable2PC protocol can use) between the Endpoint reference for App2's Protocol service and the Endpoint reference for Coordinator B's Protocol service Pb.
8. This registration causes Coordinator B to forward the registration on to Coordinator A's Registration service RSa.
9. RSa sends back to Coordinator B a RegisterResponse element that contains an EndpointReference to Protocol Service Pa. This forms a logical connection between the EndpointReferences for Pa and Pb that the Durable2PC protocol can use.

10.3.7 Addressing requirements for WS-Coordination message types

CreateCoordinationContext and Register messages:

- *Must* include a wsa:MessageID header
- *Must* include a wsa:ReplyTo header

CreateCoordinationContextResponse and RegisterResponse messages:

- *Must* include a wsa:RelatesTo header that specifies the MessageID from the corresponding request message

10.4 WS-Atomic Transaction

As we discussed in Section 10.3, "WS-Coordination" on page 397, the WS-Coordination specification defines an extensible framework for defining coordination types. The WS-Atomic Transaction specification builds on

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000437

WS-Coordination by providing the definition of the atomic transaction coordination type.

Atomic transactions have an *all-or-nothing* property. The actions taken prior to commit are only tentative (that is, not persistent and not visible to other activities). When an application finishes, it requests the Coordinator to determine the outcome for the transaction. The Coordinator determines if there were any processing failures by asking the participants to vote. If the participants all vote that they were able to execute successfully, the Coordinator commits all actions taken. If a participant votes that it needs to abort or a participant does not respond at all, the Coordinator aborts all actions taken. Commit makes the tentative actions visible to other transactions. Abort makes the tentative actions appear as if the actions never happened.

IBM, Microsoft, and BEA published the *Web Services- Atomic Transaction (WS - Atomic Transaction)* specification in September, 2003; they updated it in November, 2004. Arjuna Technologies Ltd., Hitachi Ltd., and IONA Technologies joined IBM, Microsoft, and BEA in publishing *Web Services - Atomic Transaction (WS - Atomic Transaction) Version 1.0* in August, 2005. You can find these at:

```
http://www-128.ibm.com/developerworks/library/specification/ws-tx
```

Although CICS TS V3.1 was developed and tested when the November, 2004, version was the current version, the nature of the differences between the November, 2004 version and the August, 2005 version is such that it may accurately be said that CICS TS V3.1 also supports the August, 2005 version.

All information items defined by the November, 2004 and August, 2005 versions are identified by the XML namespace URI:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat
```

We associate the namespace prefix `wsat` with this namespace by using the attribute:

```
xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
```

The WS-AT specification defines the following protocols for atomic transactions:

- Completion
- Volatile Two-Phase Commit
- Durable Two-Phase Commit

Document ID: [illegible]
Confidentiality: [illegible]



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000438

10.4.1 Completion protocol

The `Completion` protocol is used by an application to tell the Coordinator to try to either commit or abort an atomic transaction. The `Completion` protocol initiates commitment processing. Based on each protocol's registered participants, the Coordinator begins with Volatile 2PC and then proceeds through Durable 2PC. After the transaction has completed, a status (Committed or Aborted) is returned to the application.

An initiator registers for this protocol by specifying the following URI for the contents of the `ProtocolIdentifier` element in the `Register` element:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat/Completion
```

Figure 10-7 illustrates the protocol abstractly.

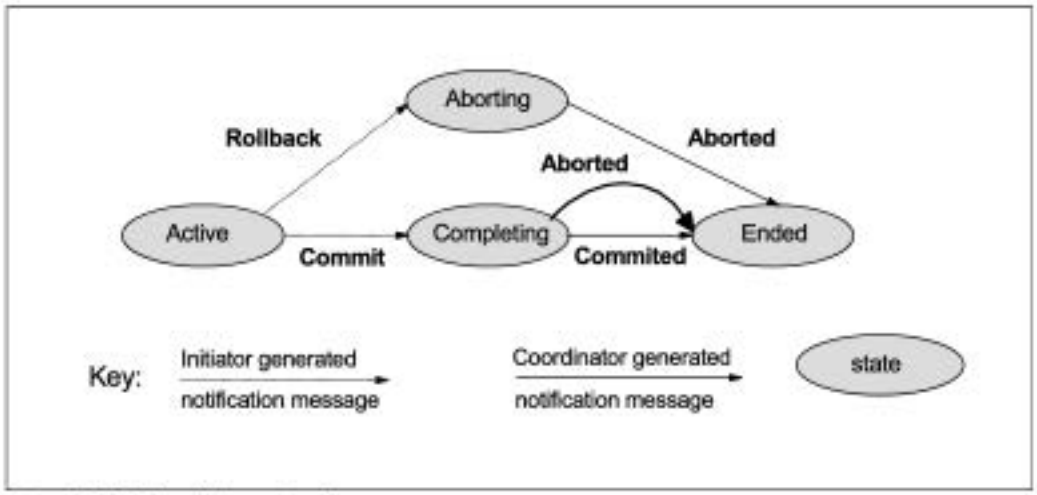


Figure 10-7 Completion protocol

The initiator generates:

- Commit

  Upon receipt of this notification, the Coordinator knows that the initiator has *completed* application processing and that it should attempt to commit the transaction.

- Rollback

  Upon receipt of this notification, the Coordinator knows that the initiator has *terminated* application processing and that it should abort the transaction.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 439/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000439

The Coordinator generates:

- Committed

  Upon receipt of this notification, the initiator knows that the Coordinator reached a decision to commit.

- Aborted

  Upon receipt of this notification, the initiator knows that the Coordinator reached a decision to abort.

10.4.2 Two-Phase Commit protocol

The Two-Phase Commit (2PC) protocol defines how multiple registered participants reach agreement on the outcome of an atomic transaction. The 2PC protocol has two variants: Volatile 2PC and Durable 2PC.

Participants managing volatile resources such as a cache should register for this protocol by using the following protocol identifier:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat/Volatile2PC
```

Participants managing durable resources such as a database should register for this protocol by using the following protocol identifier:

```
http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
```

Note: When CICS TS V3.1 is a participant in an atomic transaction, it always requests the Durable2PC protocol when it sends a Register request. When CICS TS V3.1 is the coordinator of an atomic transaction, it will tolerate a Register request for Volatile2PC but it will treat it as a Durable2PC request.

After receiving a **`Commit`** notification in the Completion protocol, the root Coordinator begins the Prepare phase of all participants registered for the Volatile 2PC protocol. All participants registered for this protocol must respond before a Prepare is issued to a participant registered for the Durable 2PC protocol. We illustrate this in Figure 10-8, where participants P1 and P3 registered for the Volatile 2PC protocol and participant P2 registered for the Durable 2PC protocol. Both P1 and P3 must respond to the **`Prepare`** notification before the Coordinator can send **`Prepare`** to P2.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 44of6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000440

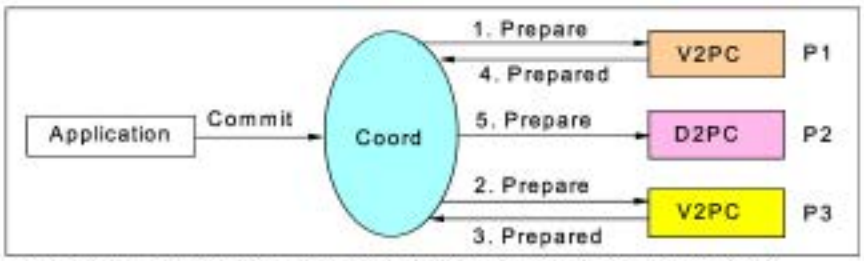


Figure 10-8 Mixture of participants registered for Durable 2PC and Volatile 2PC

Upon successfully completing the prepare phase for Volatile 2PC participants, the root Coordinator begins the Prepare phase for Durable 2PC participants. All participants registered for this protocol must respond `Prepared` or `ReadOnly` before a Commit notification is issued to a participant registered for either protocol. A volatile participant is not guaranteed to receive a notification of the transaction's outcome.

Figure 10-9 illustrates the 2PC protocol abstractly.

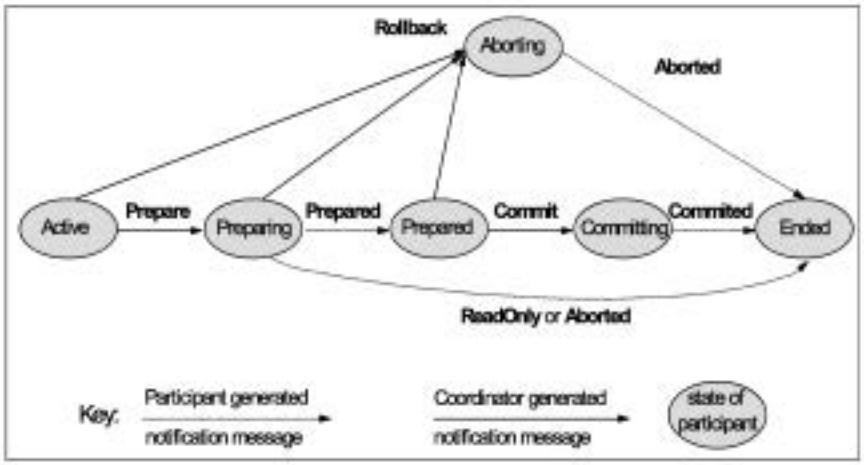


Figure 10-9 Two-Phase Commit protocol

The Coordinator generates:

- **`Prepare`**

  Upon receipt of this notification, the participant should enter phase 1 and vote on the outcome of the transaction.

  - If the participant has already voted, it should resend the same vote.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 441/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000441

– If the participant does not know of the transaction, it must vote to abort.

- **Rollback**

  Upon receipt of this notification, the participant should abort and forget the transaction. This notification can be sent in either phase 1 or phase 2. Once sent, the Coordinator may forget all knowledge of this transaction.

- **Commit**

  Upon receipt of this notification, the participant should commit the transaction. This notification can only be sent after phase 1, and if the participant voted to commit. If the participant does not know of the transaction, it must send a **Committed** notification to the Coordinator.

The participant generates:

- **Prepared**

  The participant is prepared and votes to commit the transaction.

- **ReadOnly**

  The participant votes to commit the transaction and has forgotten the transaction. The participant does not wish to participate in phase two.

  Suppose, for example, that the participant received an account number that it could not match to an entry in a database. It might return an error to the requesting application, but, having registered as a participant in the atomic transaction, it would then go on to be coordinated during 2PC processing. When the Coordinator sends **Prepare**, the participant replies **ReadOnly** and then terminates without waiting for the **Commit**. The Coordinator, on receipt of the **ReadOnly**, would then delete its own record of the interaction with the participant and would not attempt to send a **Commit** to it.

- **Aborted**

  The participant has aborted, and forgotten, the transaction.

- **Committed**

  The participant has committed the transaction. The Coordinator may safely forget that participant.

- **Replay**

  The participant has suffered a recoverable failure. The Coordinator should resend the last appropriate protocol notification.

10.4.3 Two applications with their own coordinators (continued)

In Section 10.3.6, "Two applications with their own coordinators" on page 406 we saw how two application services (App1 and App2) with their own coordinators

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000442

(Coordinator A and Coordinator B) used the WS-Coordination specification to interact as an activity propagated between them. Figure 10-10 shows how they use the WS-AT specification to complete their global unit of work.

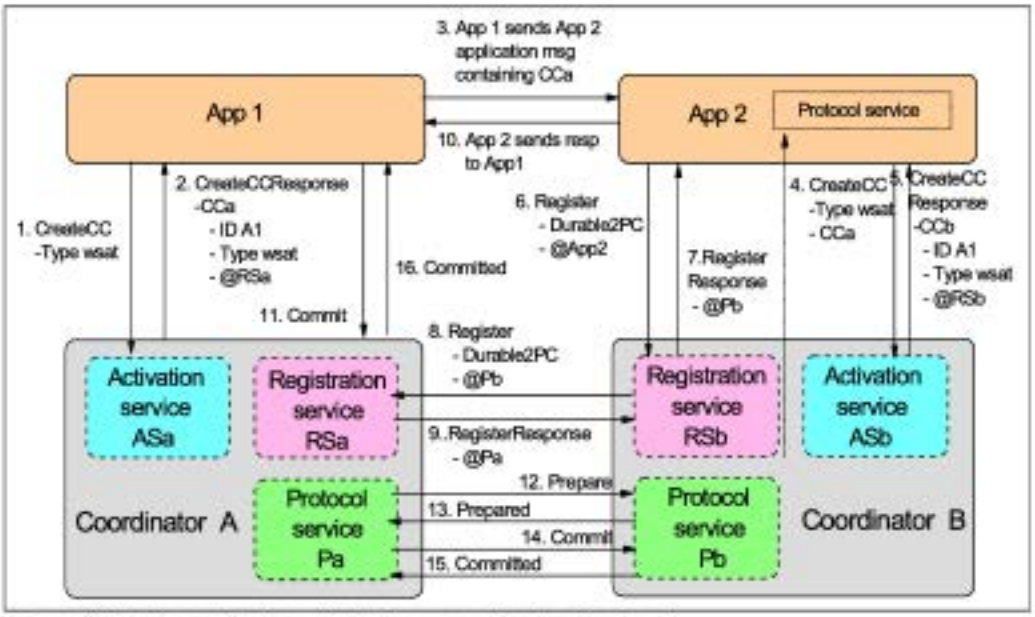


Figure 10-10 Two applications with their own coordinators (continued)

After completing its work, App2 sends its response back to App1 (step 10 in Figure 10-10). When App1 completes its work, its sends a **Commit** message to Coordinator A (step 11). This causes Coordinator A's Durable 2PC Protocol service Pa to send a **Prepare** notification to Coordinator B's Durable 2PC Protocol service Pb (step 12). We assume that Pb responds with a **Prepared** notification. If Pa encounters no other errors, it makes the decision to commit the atomic transaction and sends a **Commit** notification to Pb (step 14). Pb returns a **Committed** notification and then completes its updates before terminating. When Coordinator A receives this notification, it commits its own updates and notifies App1 of the outcome (step 16).

10.4.4 Addressing requirements for WS-AT message types

The messages defined in the WS-AT specification are *notification* messages; that is, they are *one-way* messages. There are two types of notification messages:

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 443/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000443

- A notification message is a *terminal* message when it indicates the end of a coordinator/participant relationship. **Committed**, **Aborted**, and **ReadOnly** are terminal messages.
- A notification message is a *non-terminal* message when it does not indicate the end of a coordinator/participant relationship. **Commit**, **Rollback**, **Prepare**, **Prepared**, and **Replay** are non-terminal messages.

Non-terminal notification messages *must* include a `wsa:ReplyTo` header.

10.4.5 CICS TS V3.1 and resynchronization processing

We have completed our discussion of the WS-AT specification. Unfortunately, the current version of the specification does not completely cover all of the issues surrounding the use of the 2PC protocol. In particular, it does not completely describe the resynchronization processing that should take place following a failure in one of the systems involved in the 2PC protocol or in the network connections that link the systems together. The only thing that the specification mentions relating to resynchronization is the **Replay** notification. Therefore, in this section we describe some aspects of how CICS TS V3.1 handles resynchronization processing for transactions that use the 2PC protocol.

Note: For the sake of brevity we do not describe all of the possible issues. For example, we do not describe:

- What happens when resync processing is driven from both sides and a *race* condition results
- What happens when a resync request fails

Network failures can result in messages not being delivered in a timely manner. System failures prevent processing altogether until a restart takes place.

Within the 2PC processing sequence there is a period of time, known as the *in-doubt* window, during which one system is unable to complete processing because it does not know what the other system has done. The distributed UOW is said to be in-doubt when:

- A participant Protocol service has replied **Prepared** in response to a **Prepare** notification, *and*
- Has written a log record of its response to signify that it has entered the in-doubt state, *and*
- Does not yet know the decision of its coordinator (to **Commit** or to **Rollback**).

Barring system or network failures, the UOW remains in-doubt until the coordinator issues either the **Commit** or **Rollback** request as a result of responses

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000444

received from all UOW participants. If a failure occurs that causes loss of connectivity between a participant and its coordinator, the UOW remains in-doubt until either:

- Recovery from the failure has taken place and synchronization can resume *or*
- The in-doubt waiting period is terminated by some built-in control mechanism, and an arbitrary (heuristic) decision is then taken (to commit or back out).

Note that while the UOW remains in-doubt, the recoverable resources that it owns remain locked.

If a system or network failure occurs during the in-doubt window, additional steps must be taken to ensure that the updates are completed in a consistent manner by both systems. This is known as *resynchronization processing*.

Previous releases of CICS provided a Recovery Manager that dealt with resynchronization processing for distributed workloads that made use of *VTAM®* networks or that used *MRO* connections. These releases dealt with failures during the in-doubt window in one of three ways:

- Automatic heuristic decision

  You could cause CICS to make an automatic heuristic decision by specifying the WAIT, WAITTIME, and ACTION attributes on a TRANSACTION definition as follows:

  - If you set the WAIT attribute to NO, then CICS took whatever action was specified on the ACTION attribute (either COMMIT or BACKOUT) *immediately*.
  - If you set the WAIT attribute to YES and the WAITTIME attribute to a non-zero value, then CICS took whatever action was specified on the ACTION attribute after waiting for the amount of time specified in WAITTIME (assuming normal recovery and resynchronization had not already taken place).

- Manual heuristic decision

  You could force an in-doubt UOW to complete by issuing a CEMT SET UOW(*uowid*) |COMMIT | BACKOUT| command or its EXEC CICS equivalent.

- Automatic resynchronization

  If you set the WAIT attribute to YES and the WAITTIME attribute to 00.00.00, the transaction would wait until it could communicate with its partner system, after which it could either explicitly request that the message it was waiting for be sent again, or it could resend the last message that it generated.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000445

CICS TS V3.1 extends the Recovery Manager for use by WS-AT workloads. CICS applications that form part of a WS-AT workload can be controlled by any of these mechanisms. However, automatic resynchronization is somewhat different for WS-AT workloads.

The principle difference arises from the fact that WS-AT processing takes place over a TCP/IP network.

- Other forms of distributed transactions make use of communication mechanisms such as VTAM, and resynchronization across a VTAM network can be triggered when the connection between a pair of systems is re-established.
- CICS does not currently support TCP/IP connections in the same way that it does its VTAM connections, and so CICS can only drive resynchronization of WS-AT requests when a region starts.

Important: CICS only drives resynchronization of WS-AT requests during a CICS region restart.

During any type of startup except an initial start, CICS reads the system log to discover any units of work that were in-doubt when the region previously shut down or failed. While reading through the log, CICS may find that it has outstanding units of work that indicate they were involved in an atomic transaction. These log records also indicate whether the UOW was acting as a coordinator or a participant.

- Coordinator

  If it is a coordinator and the log record indicates that the UOW was waiting for a **`Committed`** or **`Aborted`** response from a participant when CICS shut down, then the UOW is reactivated (unshunted) and sends out its decision message (**`Commit`** or **`Rollback`**) to the participant identified in the log record.

  - If a response is received, then the UOW completes its processing and terminates.
  - If a response is not received before the coordination UOW times out, then CICS shunts the UOW (moves it aside for processing later on). The UOW then persists until another resynchronization attempt takes place or until someone manually forces it to complete. (The coordination UOW times out after 30 seconds, a value set internally by CICS).

- Participant

  If it is a participant and the log record indicates that the UOW had voted in response to a **`Prepare`** message and was waiting for a **`Commit`** or **`Rollback`** decision from its coordinator when communication was lost, then the UOW is

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 446/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000446

reactivated, sends a **Replay** message to its coordinator, and once again waits for the decision message to arrive.

If the decision message is then received, the participant acts on it and sends a **Committed** or **Aborted** message back to the coordinator before terminating.

- If the decision message does not arrive before the participant UOW times out, then CICS shunts the participant UOW. The UOW then persists until another resynchronization attempt takes place or until someone manually forces it to complete.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000447

11

Enabling atomic transactions

We begin this chapter by showing you how to enable atomic transactions in CICS. We start with the simple case where we have a service requester application running in CICS AOR1 invoking a service provider application running in CICS AOR2. Then we move to the more elaborate case where two CICSplexes are working together, one acting as service requester and the other as service provider. We conclude the chapter by showing you how to enable atomic transactions in WebSphere Application Server.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000448

11.1 Enabling atomic transactions in CICS

We recognize that it is not likely that many customers will choose to use WS-AT for workloads distributed entirely within CICS. More typically, CICS would participate in an atomic transaction as a Web service provider with the service requester running under the control of another product such as WebSphere Application Server V6 or later. Customers might also use a CICS transaction acting as a Web service requester to participate in an atomic transaction with a service provider running under the control of another product.

Nevertheless, we start with the special case of one CICS region acting as a service requester participating in an atomic transaction with another CICS region acting as a service provider. This will illustrate the capacity of CICS to undertake both the role of a coordinator of an atomic transaction and the role of a participant in an atomic transaction.

11.1.1 CICS to CICS configuration

Figure 11-1 shows two CICS regions: AOR1 and AOR2. A service requester application running in AOR1 invokes a service provider application running in AOR2.

In AOR1 the request passes through a pipeline that contains a CICS-provided message handler module (either DFHPISN1, if SOAP 1.1 is being used; or DFHPISN2, if SOAP 1.2 is being used). DFHPISNx invokes the CICS-provided header processing program DFHWSATH. DFHWSATH adds a SOAP header containing a CoordinationContext to each message that it sends out.

In AOR2 the request passes through a pipeline that supports the Web service that AOR1's application is calling and also invokes DFHPISNx. DFHPISNx invokes the header processing program DFHWSATH when it detects a SOAP header that contains a `CoordinationContext`.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000449

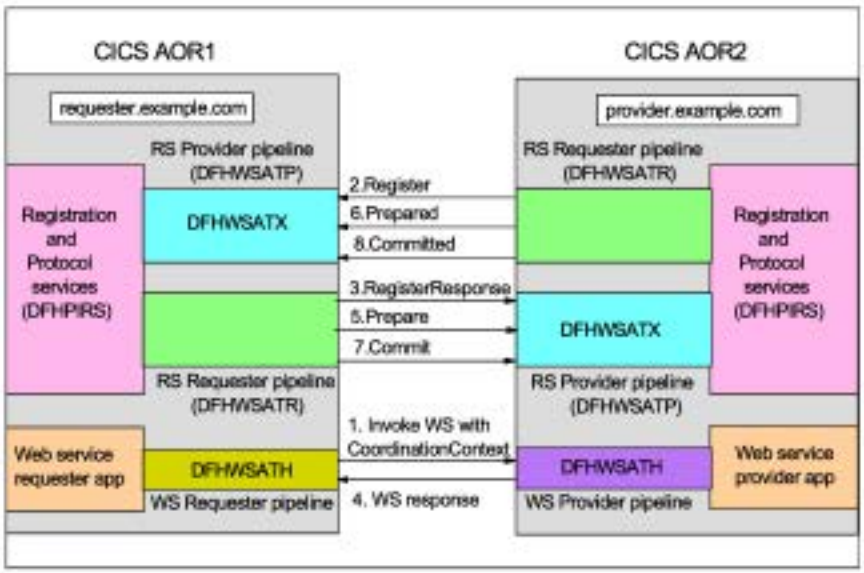


Figure 11-1 The special case: CICS to CICS

Both regions have a requester pipeline named DFHWSATR and a provider pipeline named DFHWSATP for registration and protocol processing. The DFHWSATP pipeline invokes the CICS-supplied message handler DFHWSATX as the last message handler in the pipeline.

For the workload we have shown, AOR1's DFHWSATP pipeline receives registration requests and protocol notifications, while its DFHWSATR pipeline sends registration responses and protocol instructions.

AOR2's DFHWSATP pipeline receives registration responses and protocol instructions, while its DFHWSATR pipeline sends registration requests and protocol notifications.

Note: The DFHWSATP pipeline acts as the registration endpoint for CICS.

CICS TS V3.1 provides a new resource group DFHWSAT to assist customers with setting up WS-AT support in CICS. The DFHWSAT group contains the resources shown in Table 11-1.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 450/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000450

Table 11-1 CICS supplied resource definitions for WS-AT

Resource	Resource name	Description
Pipeline	DFHWSATP	Registration services provider PIPELINE
Pipeline	DFHWSATR	Registration services requester PIPELINE
Urimap	DFHWSJPI	URIMAP used by the Registration services provider
Program	DFHPIRS	Registration and protocol services program
Program	DFHWSATH	SOAP header processing program
Program	DFHWSATR	Registration and coordination services handler program
Program	DFHWSATX	CICS message handler program

Since DFHLIST does not include the DFHWSAT group and you cannot add the DFHWSAT group to DFHLIST, specifying DFHLIST in the system initialization table GRPLIST parameter will not cause CICS to install DFHWSAT automatically during an initial start.

Figure 11-2 shows the definition of the DFHWSATP PIPELINE resource.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000451

```
OBJECT CHARACTERISTICS                                  CICS RELEASE = 0640
 CEDA  View Pipeline( DFHWSATP )
  Pipeline        : DFHWSATP
  Group           : DFHWSAT
  Description     :
  STatus          : Enabled            Enabled | Disabled
  Configfile      : /usr/lpp/cicsts/cicsts31/pipeline/configs/registrationservi
  (Mixed Case)    : ceePROV.xml
                  :
                  :
                  :
  Shelf           : /var/cicsts/
  (Mixed Case)    :
                  :
                  :
                  :
  wsdir           :
  (Mixed Case)    :
                  :
                                                  SYSID=3C3 APPLID=WAPC1JCC
```

Figure 11-2 Definition of the DFHWSATP PIPELINE resource

Note that the definition contains the fully-qualified name of the pipeline configuration file. If you install the CICS-supplied registrationservicePROV.xml configuration file in a different directory when you install CICS (as we did), then you must make a copy of the online DFHWSAT group, change the definition of the DFHWSATP pipeline, and install the modified group.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000452

Tip: If you attempt to modify the definition of the DFHWSATP pipeline in group DFHWSAT, you will get the message "`Unable to perform operation: DFHWSAT is IBM protected.`"

If you add the DFHWSAT group to your startup list and then attempt to override the definition of DFHWSATP that is provided in the DFHWSAT group by adding a modified DFHWSATP definition that appears later in the startup list, then you will get message DFHAM4892 W indicating that the install of the second group completed with errors.

If you copy only the DFHWSATP, DFHWSATR, and DFHRSURI definitions to a new group and try to let program autoinstall automatically install the definitions of the programs DFHWSATH, DFHWSATR, DFHWSATX, and DFHPIRS, you may also have problems. These four programs need access to containers which use CICS-key storage, and therefore they must run with EXECKEY(CICS) unless storage protection is turned off. You would have a problem if the model for the program you use for program autoinstall does not specify EXECKEY(CICS).

Example 11-1 shows the contents of the registrationservicePROV.xml file. This configuration file defines a pipeline that contains only one message handler program, DFHWSATX.

Example 11-1 The registrationservicePROV.xml pipeline configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <service>
    <terminal_handler>
      <handler>
        <program>DFHWSATX</program>
        <handler_parameter_list/>
      </handler>
    </terminal_handler>
  </service>
  <service_parameter_list/>
</provider_pipeline>
```

Figure 11-3 shows the definition of the DFHWSATR PIPELINE resource.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000453

```
OBJECT CHARACTERISTICS                                    CICS RELEASE = 0640
 CEDA  View PIpeline( DFHWSATR )
  PIpeline       : DFHWSATR
  Group          : DFHWSAT
  Description    :
  STatus         : Enabled            Enabled | Disabled
  Configfile     : /usr/lpp/cicsts/cicsts31/pipeline/configs/registrationserv
  (Mixed Case)   : iceREQ.xml
                 :
                 :
                 :
  SHelf          : /var/cicsts/
  (Mixed Case)   :
                 :
                 :
                 :
  Wsdir          :
  (Mixed Case)   :
                 :

                                                  SYSID=T3C1 APPLID=A6POT3C1
```

Figure 11-3 Definition of the DFHWSATR PIPELINE resource

Example 11-2 shows the contents of the registrationserviceREQ.xml file. This configuration file defines a pipeline that does not contain any message handlers.

Example 11-2 The registrationserviceREQ.xml pipeline configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
    xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
    requester.xsd">
</requester_pipeline>
```

Important: We recommend that you do not modify the registrationserviceREQ.xml pipeline configuration file or the registrationservicePROV.xml pipeline configuration file. If you modify one of these files, you might inadvertently alter the flow of the registration or protocol service messages and thereby affect the integrity of your data.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000454

Figure 11-4 shows the definition of the DFHRSURI URIMAP resource.

```
 OBJECT CHARACTERISTICS                                  CICS RELEASE = 0640
  CEDA View Urimap( DFHRSURI )
   Urimap          : DFHRSURI
   Group           : DFHWSAT
   Description     :
   Status          : Enabled            Enabled | Disabled
   USAge           : Pipeline           Server | Client | Pipeline
  UNIVERSAL RESOURCE IDENTIFICA
   SCheme          : HTTP               HTTP | HTTPS
   HOST            : *
   (Lower Case)    :
   PATH            : /cicswsat/RegistrationService
   (Mixed Case)    :
                   :
                   :
                   :
  ASSOCIATED CICS RESOURCES
   TCPipeservice   :
   Analyzer        : No                 No | Yes
   CONverter       :
   TRANsaction     : CPIH
   PROgram         :
   PIpeline        : DFHWSATP
   WEBservice      :                                          (Mixed Case)
  SECURITY ATTRIBUTES
   USEIrc          :
   Ciphers         :
```

Figure 11-4 Definition of the DFHRSURI URIMAP resource

When the CICS-supplied CWXN transaction finds that the URI in an HTTP request matches the PATH attribute of the DFHRSURI URIMAP definition, it uses the PIPELINE attribute of that definition to get the name of the PIPELINE definition that it will use to process the incoming request. As Figure 11-4 shows, this is the DFHWSATP pipeline. As we have already seen, the DFHWSATP PIPELINE definition specifies that CICS should use registrationservicePROV.xml as the pipeline configuration file. As we have also already seen, this configuration file defines a pipeline that contains only one message handler program, DFHWSATX. Thus when the URI in an HTTP request contains /cicswsat/RegistrationService, CICS invokes the DFHWSATX message handler.

This raises the question: What causes CICS to receive an HTTP request whose URI field contains /cicswsat/RegistrationService?

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000455

The answer is that the pipeline configuration file for the service requestor application (running in CICS AOR1) must specify:

- One of the CICS-provided SOAP message handlers (cics_soap_1.1_handler or cics_soap_1.2_handler)
- The mandatory invocation of the DFHWSATH header processing program to add a CoordinationContext header to the SOAP request
- A <registration_service_endpoint> element within a <service_parameter_list>

This is shown in Example 11-3

Example 11-3 Service requester pipeline configuration file which supports WS-AT

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
      requester.xsd">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>DFHWSATH</program_name>
          <namespace>
             http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </service_handler_list>
  </service>
  <service_parameter_list>
    <registration_service_endpoint>
       http://requester-example.com:3207/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</requester_pipeline>
```

The <registration_service_endpoint> element contains the address of the Registration service endpoint that runs in the requesting CICS region (AOR1). The path component of this address matches the PATH attribute defined in the DFHRSURI URIMAP resource definition of AOR1. Participant Web services should send Register requests and Prepared and Committed (or Aborted) notifications to this address.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000456

In the service requester pipeline in Example 11-3

- Since the <mandatory> element contains True, the pipeline will flow a CoordinationContext with the message.
- If you change the <mandatory> element to false or remove DFHWSATH from the pipeline, the pipeline will not flow a CoordinationContext with the message.

The pipeline configuration file (Example 11-4) for the service *provider* application must specify:

- One of the CICS-provided SOAP message handlers (cics soap 1.1 handler or cics soap 1.2 handler)
- Invocation of the DFHWSATH header processing program whenever the SOAP message contains a CoordinationContext header
- A <registration_service_endpoint> element within a <service_parameter_list>

Example 11-4 Service provider pipeline configuration file that supports WS-AT

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
      provider.xsd ">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>DFHWSATH</program_name>
          <namespace>
             http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>false</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
  <service_parameter_list>
    <registration_service_endpoint>
    http://provider.example.com:3207/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</provider_pipeline>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000457

This time the <registration_service_epr> element contains the address of the Registration service endpoint that runs in the provider CICS region. The Coordinator should send RegisterResponse messages and Prepare and Commit (or Abort) notifications to this address.

In the service provider pipeline in Example 11-4:

- The pipeline will accept flows with a CoordinationContext, and such flows will be treated as part of a WS-AT transaction
- Since the <mandatory> element contains False, the pipeline will also accept messages *without* a CoordinationContext but they will *not* be part of any WS-AT transaction
- If you change the <mandatory> element to True, the pipeline will *require* that a CoordinationContext flow with the message. A Fault will be raised if a requestor attempts to use the service without a CoordinationContext.
- If you remove DFHWSATH from the pipeline, the pipeline will raise a mustUnderstand fault when a CoordinationContext arrives with mustUnderstand set to True

We conclude this section with a few remarks on the function provided by two programs in the CICS-supplied DFHWSAT group.

DFHPIRS

DFHPIRS is the central component of CICS support for WS-AT. It provides the bulk of the function that is needed for registration and protocol processing. The actions that it can be called to service include the following: Register, RegisterResponse, Prepare, Prepared, Aborted, ReadOnly, Commit, Rollback, Committed, and Replay.

If DFHPIRS encounters an unrecoverable error, it abends with an abend code of APIO.

DFHWSATH

When included in the configuration file for a Web service *requester* pipeline, DFHWSATH controls the processing that causes a CoordinationContext to be created and added to the SOAP message before it is sent. To do this DFHWSATH calls DFHPIAT which finds the local netname, the unit-of-work ID (UOWID), and the value of the DTIMOUT attribute on the TRANSACTION definition for the transaction ID under which the requesting application is running. DFHPIAT also finds the endpoint address of the Registration service for the requesting region. From all of this information DFHPIAT creates a CoordinationContext.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000458

When included in the configuration file for a Web service *provider* pipeline, this program is invoked if the CICS-supplied SOAP message handler detects a `CoordinationContext` header in a message. When called, DFHWSATH controls the processing that extracts data from the `CoordinationContext` header.

If DFHWSATH encounters an unrecoverable error, it abends with an APIP abend, which causes the pipeline manager to terminate the processing of the current request.

11.1.2 More elaborate CICS to CICS configuration

In the simple CICS to CICS configuration shown in 11.1.1, "CICS to CICS configuration" on page 420, all the requester registration endpoints reside in the same region as the service requester application, and all the provider registration endpoints reside in the same region as the service provider application. This does not have to be the case, and a number of different configurations are possible, including the one shown in Figure 11-5.

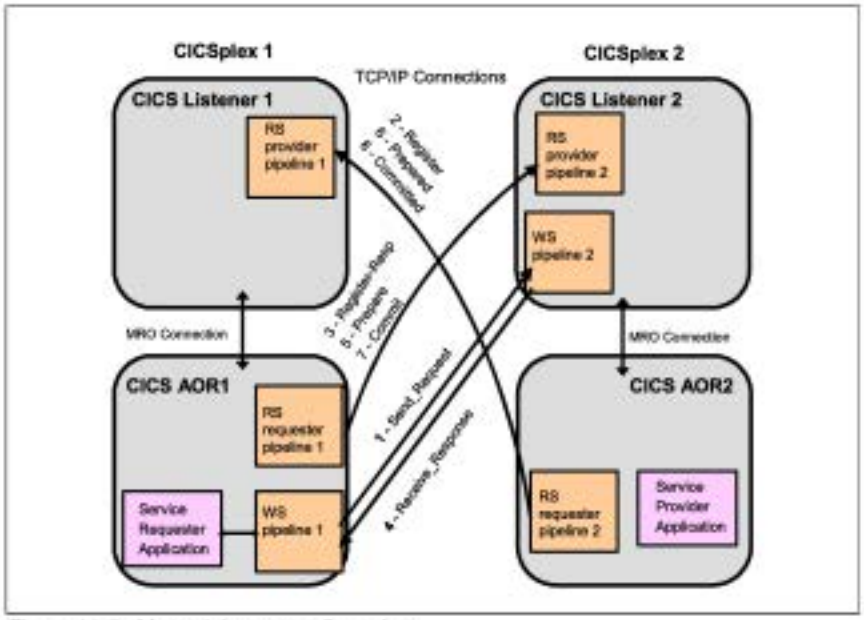


Figure 11-5 More elaborate configuration

The service requester pipeline (WS pipeline 1) must run in the same region as the service requester application that it supports. However, the service provider pipeline (WS pipeline 2) does not have to run in the same region as the service

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 459/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000459

provider application that it manages. Instead, these applications are eligible for dynamic routing as described in 3.4, "Configuring for high availability" on page 101. Similarly, Registration services provider pipelines, such as RS provider pipeline 1 and RS provider pipeline 2 in Figure 11-5, do not have to be located in the same region as the applications that they service.

In the case of RS provider pipeline 1, the requests arriving there are for registration with the service requester application running in AOR1, and protocol response messages that refer to this application's role in coordinating the atomic transaction. The RS provider pipeline 2 receives registration response messages and protocol notification messages intended for the service provider application running in AOR2.

Since eventually these messages must be processed in the same region as the service provider application, the WS-AT implementation in CICS TS makes use of *RequestStreams* to pass these messages from one region to another. RequestStreams are dependent on MRO connections when they exchange information between CICS regions.

Note: If a Registration Service provider pipeline is configured in a different region than the application that it services, then an MRO connection is required between the CICS regions.

The use of RequestStreams means that individual Registration services provider pipelines can be used by different applications, and those applications do not have to reside in the same region. A set of cloned AORs can share a common Registration services provider pipeline that is located in a specific WS-AT CICS region. Work can then be distributed across the AORs and each of the WS-AT messages, arriving at the Registration services provider pipeline, contains sufficient information to allow it to be forwarded to the correct AOR.

Tip: A set of cloned AORs can share a common RS provider pipeline located in a specific WS-AT region, thus reducing the number of pipeline configurations required. In this configuration, it is recommended that you do not run applications in the specific WS-AT region and that you create clones for improved failover and availability.

Figure 11-5 shows the Registration services requester pipelines are in the same region as the applications that they service. RS requester pipeline 1 services the service requester application in AOR1, and RS requester pipeline 2 services the service provider application in AOR2. This configuration cannot be changed because CICS requires that each WS-AT message must be dispatched from the same region as the application that it relates to. This means that each AOR in a

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000460

cloned set of AORs used to support a common Web services workload, must have it's own Registration services requester pipeline configured.

Note: A Registration services requester pipeline must be configured in each AOR that hosts applications which participate in atomic transactions.

11.2 Enabling atomic transactions in WebSphere

WebSphere Application Server Version 6 implements the WS-AT specification. A J2EE application programmer demarcates a global transaction in a program by using the Java Transaction API (JTA) `UserTransaction` interface as shown in Example 11-5.

Example 11-5 Demarcating a global transaction using the JTA

```
UserTransaction userTransaction = null;
try {
      InitialContext context = new InitialContext();
      userTransaction = (UserTransaction)
                              context.lookup("java:comp/UserTransaction");
      userTransaction.begin();

      // insert record into database
        .
        .
        .
      // commit
      userTransaction.commit();
    } catch (java.rmi.RemoteException re) {
        try {
           userTransaction.rollback();
        }
        .....
     }
```

If a Web service request is made by an application component running under a global transaction, WebSphere Application Server implicitly propagates a `CoordinationContext` to the target Web service if the appropriate application deployment descriptors have been specified.

If WebSphere Application Server is the system hosting the target endpoint for a Web service request that contains a WS-AT `CoordinationContext`, WebSphere automatically establishes a subordinate JTA transaction in the target run-time environment that becomes the transactional context under which the target Web service application executes.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net

Page 461/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000461

Application developers do not have to explicitly register WS-AT participants. The WebSphere Application Server runtime takes responsibility for the registration of WS-AT participants. At transaction completion time, all WS-AT participants are atomically coordinated by the WebSphere Application Server transaction service.

There are no specific development tasks required for Web service applications to take advantage of WS-AT; however, there are some application deployment descriptors that have to be set appropriately.

- In a Web module that invokes a Web service, specify Send Web Services Atomic Transaction on requests to propagate the transaction to the target Web service.

  See "Change the deployment descriptor" on page 450 for information on how we enabled our Web application to send WS-AT SOAP headers in requests to a CICS service provider application.

- In a Web module that implements a Web service, specify Execute using Web Services Atomic Transaction on incoming requests to run under a received client transaction context.
- In an EJB module that invokes a Web service, specify Use Web Services Atomic Transaction to propagate the EJB transaction to the target Web service.
- In an EJB module, bean methods must be specified with transaction type Required, which is the default, to participate in a global atomic transaction.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000462

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000463

12

Transaction scenarios

In this chapter we show different scenarios that demonstrate how you can synchronize resource updates using the WS-Atomic Transaction support in CICS and WebSphere Application Server.

We start with an explanation of the scenarios that we set out to test, and how we prepared the system and the settings that we used for the configuration of our system.

We then cover in detail the changes that we made to the CICS pipeline configuration and WebSphere Application Server to enable Web services transactional integration.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000464

12.1 Introduction to our scenarios

Figure 12-1 shows three different atomic transaction scenarios that we considered testing.

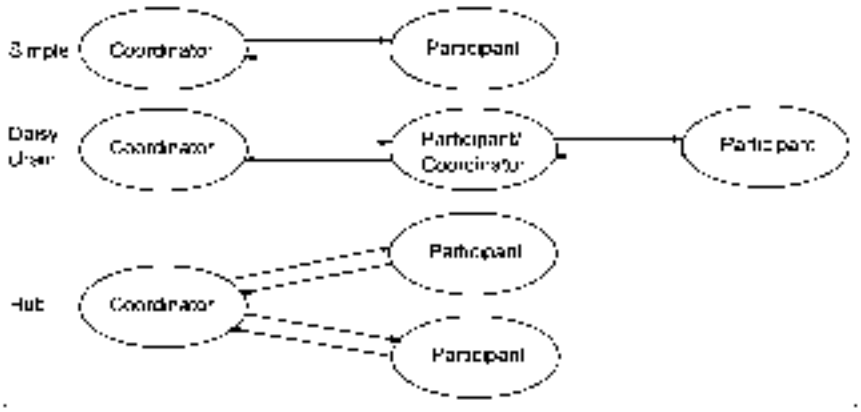


Figure 12-1 Three possible atomic transaction scenarios

The scenario at the top is the simple case where a coordinator controls a single participant. Both parties may make recoverable updates. Another possibility is that only the participant makes recoverable updates. A third possibility is that the participant does nothing recoverable; in this case it will send a ReadOnly notification when it is coordinated.

The second scenario is often referred to as a *daisy chain*. Here there is a primary (or root) coordinator that invokes a Web service, and the invoked Web service then invokes a second Web service. The online atomic transaction is controlled by the primary coordinator. The middle system takes on the role of a coordinator *and* the role of a participant at different times in its life cycle. When the primary coordinator instructs it during transaction termination, this system acts as a participant. However, before it responds to the primary coordinator, it then takes on the role of coordinator of its own participant.

The third scenario is a *hub* configuration. A single coordinator invokes one Web service and then another. It then coordinates them together.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000465

You can probably think of many other scenarios. In this chapter we more closely look at two of these scenarios:

- The simple scenario in which WebSphere Application Server V6.0 is the coordinator and CICS TS V3.1 is the participant. See "The simple atomic transaction scenario" on page 439.
- The daisy chain scenario in which CICS TS V3.1 is both a coordinator and a participant. See 12.3, "The daisy chain atomic transaction scenario" on page 482.

12.1.1 Software checklist

Table 12-1 shows the software we used in the scenarios described in this chapter.

Table 12-1 Software used in the atomic transaction scenarios

Windows	z/OS
Windows 2000 SP4	z/OS V1.6
IBM WebSphere Application Server - ND V6.0.0.2	CICS Transaction Server V3.1
Internet Explorer V6.0	
DB2 V8.1.7.445	
User-supplied programs: - CatalogAtomic.ear A modified service requester application used for the WS-AT scenarios - DispatchAtomic.ear A modified service provider application used for the WS-AT scenarios	User-supplied programs: - SNIFFER (message handler program written in COBOL) - WSATHND (header processing program written in C)

Important: You should install the fix for APAR PK16509 if you are using a later version of WebSphere than IBM WebSphere Application Server - Network Deployment V6.0.0.2.

12.1.2 Definition checklist

Table 12-2 shows the definitions we used in the scenarios described in this chapter.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 466/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000466

Table 12-2 Definitions used in the atomic transaction scenarios

Value	CICS TS	WebSphere Application Server
IP name	mvsg3.mop.ibm.com	cam21-pc1[illegible].mop.ibm.com
IP address	9.[illegible]0.193.167	9.[illegible]0.199.238
Job name	CWST3C1	
APPLID	A6POT3C1	
SIT parameter	SEC=NO (see APAR PK10840)	
TCPIPSERVICE definition	T3C1	
PORT attribute on T3C1 TCPIPSERVICE definition	15351	
FILE definition for sample catalog VSAM file	EXMPCAT	
RECOVERY attribute on EXMPCAT FILE definition	BACKOUTONLY	
Web service provider PIPELINE definition	PIPE1	
CONFIGFILE attribute on PIPE1 PIPELINE definition	/CICSTS31/CWS1/config/ITSO_7206_wsat_soap11provider.xml	
Web service requester PIPELINE definition	PIPE2	
CONFIGFILE attribute on PIPE2 PIPELINE definition	/CICSTS31/CWS1/config/ITSO_7206_wsat_soap11requester.xml	
RDO group containing copy of DFHWSAT group	CTS3100	
Registration Service provider PIPELINE	DFHWSATP	
CONFIGFILE attribute on DFHWSATP PIPELINE definition	/CICSTS31/CWS1/config/ITSO_7206_wsat_registrationservicePROV.xml	
Registration Service requester PIPELINE	DFHWSATR	

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000467

Value	CICS TS	WebSphere Application Server
CONFIGFILE attribute on DFHWSATR PIPELINE definition	/CIWS/T3C1/config /ITSO_7206_wsat_registrationserviceREQ.xml	

Important: As we write this book, APAR PK10849 is open. It reports that if you attempt to use WS-AT in a CICS region that is running with the SIT parameter SEC=YES, then during the RegisterResponse step of coordination RACF will issue messages ICH408I and IRR012I and CICS will issue message DFHPI0002. Therefore, we ran our region with SEC=NO.

12.2 The simple atomic transaction scenario

The sample Catalog application provided with CICS TS V3.1 provides three Web services:

- inquireSingle
- inquireCatalog
- placeOrder

Only the placeOrder Web service *updates* the VSAM file that contains the information about the company's products. Therefore, we naturally decided that the service requester running in WebSphere Application Server should invoke the placeOrder Web service in our WS-AT scenario.

We modified the service requester application (see "Installing the service requester" on page 87) to create a global transaction and to update a DB2 table. We call the new service requester application *AtomicClient*.

Note: Rather than using JDBC™ to update a DB2 table, we could have chosen to use another J2EE connector such as the JCA or JMS. Updates to resources accessed by these connectors can be synchronized with Web service requests in the same way.

To be more specific, we create the table ITSO.ORDER in DB2. Before calling the placeOrder Web service, the AtomicClient inserts a row in this table so that we have a log of all of the orders placed through our application. We can now have a global transaction that updates two resources: a DB2 table in the Windows environment and a VSAM file in the z/OS environment. Figure 12-2 shows the

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 468/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000468

sequence of events in AtomicClient as it begins a global transaction, updates a DB2 database, calls the placeOrder Web service, and then either commits or rolls back the updates.

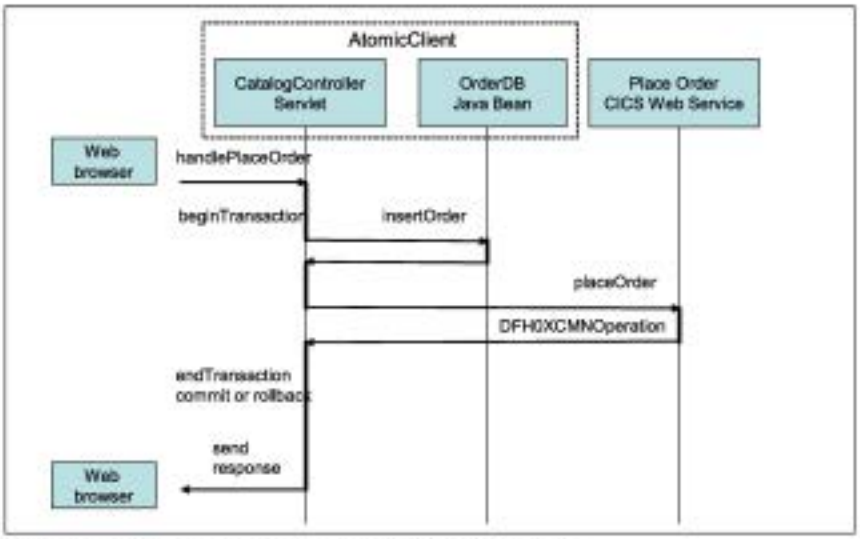


Figure 12-2 Simple atomic transaction sequence of events

Figure 12-3 shows a more global view of the sequence of events:

- The user uses his Web browser to invoke the AtomicClient that runs in WebSphere Application Server.
- The AtomicClient updates the ITSO.ORDER table in DB2.
- WebSphere Application Server sends the insertOrder SOAP message containing the order to CICS.
- CICS uses Web service provider PIPELINE definition PIPE1 to process the SOAP message. PIPE1 contains our SNIFFER program and the CICS-supplied SOAP 1.1 message handler DFHPISN1.
- DFHPISN1 links to the CICS-supplied header processing program DFHWSATH.
- CICS converts the SOAP message to a COMMAREA for the sample catalog manager program, DFH0XCMN.
- DFH0XCMN passes the data in the COMMAREA to the sample catalog program DFH0XVDS, which updates the recoverable VSAM file.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 469/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000469

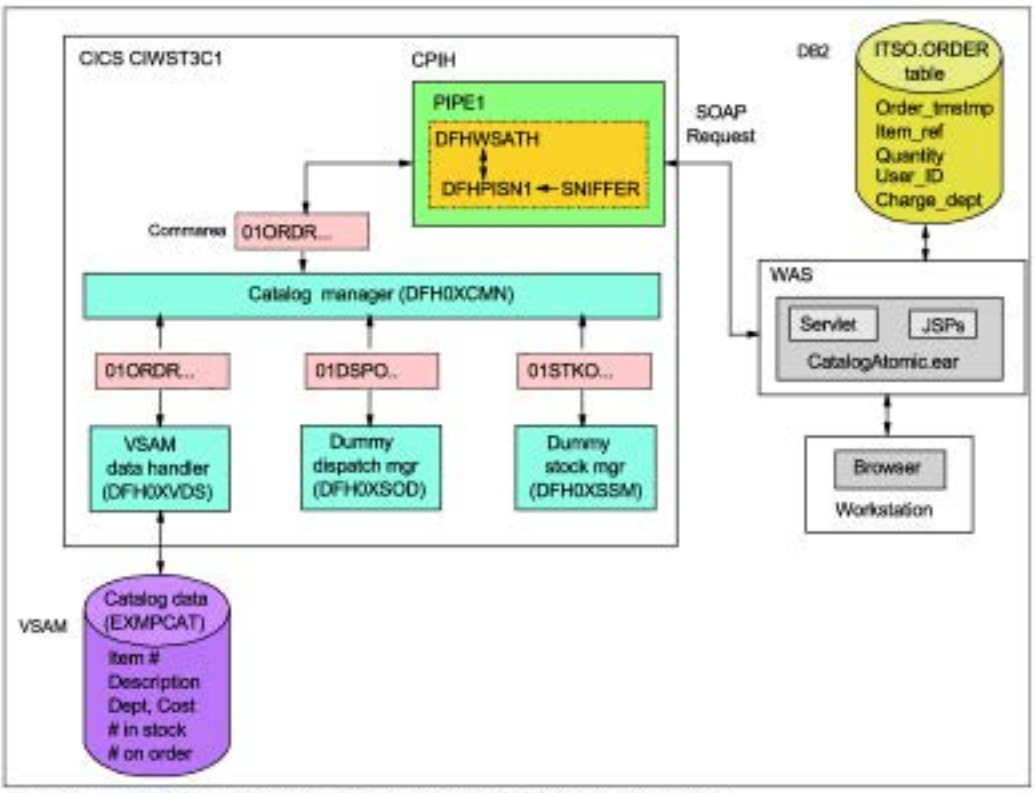


Figure 12-3 WebSphere as service requester and CICS as service provider

In the following sections we describe how we set up CICS for this scenario, how we created the AtomicClient and the ITSO.ORDER table, and the results of testing this scenario.

12.2.1 Setting up CICS for the simple scenario

To set up CICS for this scenario we performed the following steps:

- Set the value of the SEC system initialization table parameter to NO; see 12.1.2, "Definition checklist" on page 437.
- Changed the value of the RECOVERY attribute of the EXMPCAT FILE definition to BACKOUTONLY so that the file is recoverable. (The EXMPCAT FILE definition defines the VSAM file that contains the catalog data for the sample application).

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
Online/Notary.net
Page 470/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000470

- Edited PIPE1's configuration file

  /CIWS/T3C1/config/ITSO_7206_wsat_soap11provider.xml

  so that it contains the XML shown in Example 12-1. This XML contains:

  - The same XML as shown in Example 11-4 on page 428, except that we changed the address in the registration_service_endpoint element to be specific to the CICS region CIWST3C1
  - XML to add the SNIFFER message handler

Example 12-1 PIPE1: /CIWS/T3C1/config/ITSO_7206_wsat_soap11provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
      provider.xsd ">
  <service>
    <service_handler_list>
      <handler>
        <program>SNIFFER</program>
        <handler_parameter_list/>
      </handler>
    </service_handler_list>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <header_program>
          <program_name>DFHWS4IH</program_name>
          <namespace>
             http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>false</mandatory>
        </header_program>
      </cics_soap_1.1_handler>
    </terminal_handler>
  </service>
  <apphandler>DFHPITP</apphandler>
  <service_parameter_list>
    <registration_service_endpoint>
     http://MVS63.mop.ibm.com:15301/cicswsat/RegistrationService
    </registration_service_endpoint>
  </service_parameter_list>
</provider_pipeline>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000471

Tip: For readability Example 12-1 shows the `registration_service_endpoint` element split across three lines. However, during our testing we found that we had to place its start tag, contents, and end tag on the *same* line.

- Copied the group DFHWSAT to the group CTS310C and added the group CTS310C to our startup list:

```
CEDA COPY GROUP(DFHWSAT) TO(CTS310C)
CEDA ADD GROUP(CTS310C) LIST(LISTT3C1)
```

- Edited DFHWSATP's configuration file

  /CIWS/T3C1/config/ITSO_7206_wsat_registrationservicePROV.xml

  so that it contained the XML shown in Example 12-2. This XML contains:

  - The same XML as shown in Example 11-1 on page 424
  - XML to add the WSATHND header processing program (see "How we monitored the exchange of WS-AT messages" on page 444)

Example 12-2 DFHWSATP: /CIWS/.../ITSO_7206_wsat_registrationservicePROV.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
  provider.xsd ">
  <service>
    <service_handler_list>
      <cics_soap_1.2_handler>
        <headerprogram>
          <program_name>WSATHND</program_name>
          <namespace>*</namespace>
          <localname>wsatHeader</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.2_handler>
    </service_handler_list>
    <terminal_handler>
      <handler>
        <program>DFHWSATX</program>
        <handler_parameter_list/>
      </handler>
    </terminal_handler>
  </service>
  <service_parameter_list/>
</provider_pipeline>
```


Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 472/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000472

- Edited DFHWSATR's configuration file

  /CIWS/T3C1/config/ITSO_7206_wsat_registrationserviceREQ.xml

  so that it contained the XML shown in Example 12-3. This XML contains

  - The same XML shown in Example 11-2 on page 425
  - XML to add the WSATHND header processing program

Example 12-3 DFHWSATR: /CIWS/.../ITSO_7206_wsat_registrationserviceREQ.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
    xmlns="http://www.ibm.com/software/htp/cics/pipeline"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
    requester.xsd">
  <service>
    <service_handler_list>
      <cics_soap_1.2_handler>
        <headerprogram>
          <program_name>WSATHND</program_name>
          <namespace></namespace>
          <localname>wsatHeader</localname>
          <mandatory>true</mandatory>
        </headerprogram>
      </cics_soap_1.2_handler>
    </service_handler_list>
  </service>
</requester_pipeline>
```

How we monitored the exchange of WS-AT messages

The user-written WSATHND header processing program writes messages to the CESO transient data queue, which normally is an extrapartition queue with ODname CEEOUT. We used WSATHND to monitor the exchange of registration and protocol service messages between WebSphere Application Server and CICS.

The message output of WSATHND depends on the values the programmer has assigned to the two variables MESSAGES_ON and FULL_MESSAGES_ON.

1. If MESSAGES_ON is set to 1, the program writes the following message:

   WSAT: REACHED HANDLER - function

   where function is the contents of the DFHFUNCTION container (RECEIVE-REQUEST, SEND-RESPONSE, SEND-REQUEST, RECEIVE_RESPONSE, PROCESS_REQUEST, NO_RESPONSE, or HANDLER-ERROR)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000473

When the function is NO-RESPONSE, then the handler is being invoked after processing a request, when there is no response to be processed.

2. If the function is not NO-RESPONSE, then:
   - If FULL_MESSAGES_ON is set to 1, WSATHND writes the following message:

     WSAT: contents of the DFHREQUEST container
   - If MESSAGES_ON is set to 1, it will write the following message:

     WSAT: ACTION: action

     where action has one of the following values: Register, RegisterResponse, Prepare, Prepared, Commit, Committed, ReadOnly, Abort, Aborted, Rollback

Thus, WSATHND provides a way to see the registration and protocol service messages that are being sent between the service requester and the service provider as they happen. We found it easier to use this program than to take a TCP/IP trace or to search for these messages in a CICS auxiliary trace. The full WSATHND program is shown in A.7, "Sample header processing program - WSATHND" on page 539.

12.2.2 Creating the AtomicClient and ITSO.ORDER table

To create the AtomicClient and the ITSO.ORDER table, we did the following:

- Created the OrderBean JavaBean that represents the order.
- Created the OrderDB JavaBean that inserts the order into the DB2 database.
- Changed the handlePlaceOrder method of the CatalogController servlet so that it creates an OrderBean and then calls OrderDB to insert it into the DB2 database as part of a global transaction.
- Changed the deployment descriptor of the Web module CatalogAtomicWeb. This is the module that invokes the Web Service.
- Created the ITSO.ORDER table.
- Created a data source.

We explain each step in detail in the following sections.

Create the OrderBean JavaBean

The OrderBean JavaBean represents the order; it has all the fields from the PlaceOrder JavaServer™ Page (JSP™) plus a timestamp that records the time when the Web browser user placed the order. Example 12-4 shows the code for the OrderBean JavaBean.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000474

Example 12-4 The OrderBean JavaBean

```
package itso.mop.objects;

import java.sql.Timestamp;

public class OrderBean {

   private Timestamp orderTmstmp;
   private short itemRef;
   private short quantity;
   private String userId;
   private String chargeDept;

// getters and setters
...
}
```

Create the OrderDB JavaBean

Example 12-5 shows the code for the OrderDB JavaBean, which inserts the order into the DB2 table.

Example 12-5 The OrderDB JavaBean

```
package itso.mop.db;

import  itso.mop.objects.OrderBean;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.sql.DataSource;

public class OrderDB {

   private static final String _JNDI_NAME = "jdbc/CACHEDS";

   public  int insertOrder(OrderBean order){
      Connection con = getConnection();
      PreparedStatement pr = null;
      int rs  = 0;
      try {
         pr = con.prepareStatement("INSERT INTO ITSO.ORDER (ORDER_TMSTMP,
ITEM_REF, QUANTITY, USER_ID, CHARGE_DEPT) VALUES(?,?,?,?,?)");
         pr.setTimestamp(1, order.getOrderTmstmp());
         pr.setInt(2, order.getItemRef());
         pr.setInt(3, order.getQuantity());
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000475

```
            pm.setString(4, order.getUserId());
            pm.setString(5, order.getChargeCost());

            rs = pm.executeUpdate();

            System.out.println("OrderDB.insertOrder() - Inserted the order in the
database!!!");
         } catch (SQLException e) {
            System.out.println("OrderDB.insertOrder()  Exception inserting the
order in the database!!");
            e.printStackTrace(System.err);
            rs = 0;
         } finally {
            try {
               if (pm != null)
                  pm.close();
               if (con != null)
                  con.close();
            } catch (Exception e) {
               e.printStackTrace(System.err);
            }
         }
         return rs;
      }

      private Connection getConnection() {
         Connection con = null;
         try {
            InitialContext ic = new InitialContext();
            DataSource ds = (DataSource) ic.lookup("JNDI_NAME");
            con = ds.getConnection();
         } catch (NamingException e) {
            e.printStackTrace(System.err);
         } catch (SQLException e) {
            e.printStackTrace(System.err);
         }
         return con;
      }
   }
```

Change the handlePlaceOrder method

In the Catalog application the CatalogController servlet handles all the requests from the Web browser. In particular, the handlePlaceOrder method of this servlet handles requests from the Web browser to place an order. We made the following changes to this method:

1. Created the OrderBean and populated all of its fields

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000476

2. Created the transaction (UserTransaction) and then began the transaction
3. Inserted the order into the DB2 database using the OrderDB JavaBean
4. Called the CICS Web service
5. Coded the method to throw a RemoteException when the "ROLLBACK" user ID enters the order (for testing transaction rollback)
6. Committed the transaction
7. Rolled back the transaction in the case of an Exception

Note: The rollback logic was added to the application for testing purposes only; normal applications would not be expected to behave in this way.

Example 12-6 shows the code for the `handlePlaceOrder` method.

Example 12-6 The handlePlaceOrder method of the CatalogController servlet

```
private void handlePlaceOrder(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException
{
   OrderDetails orderDetails = new OrderDetails();
   try
   {
      // Retrieve the data from the submitted form
      orderDetails.setItemRefNumber(
short.parseShort(request.getParameter("itemRef")));
      orderDetails.setQuantityRequired(Short.parseShort(
         request.getParameter("quantity")));
      orderDetails.setUserId(request.getParameter("userName"));
      orderDetails.setChargeDepartment(request.getParameter("deptName"));
   }
   catch(Exception e)
   {
      handleError(request, response, "Input data format incorrect, please try
again");
      return;
   }

   UserTransaction userTransaction = null;
   try
   {
      //1. Create the order bean and populate all of its fields
      System.out.println("CatalogController.handlePlaceOrder() - creating the
order");
      OrderBean order = new OrderBean();
      order.setOrderTmstmp(new Timestamp(Calendar.getInstance().
```


Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000477

```
            getTime().getTime()));
        order.setItemRef(orderDetails.getItemRefNumber());
        order.setQuantity(orderDetails.getQuantityRequired());
        order.setUserId(orderDetails.getUserId());
        order.setChargeDept(orderDetails.getChargeDepartment());

        // 2. Create and begin the Transaction
        InitialContext context = new InitialContext();
        userTransaction = (UserTransaction)
                        context.lookup("java:comp/UserTransaction");
        System.out.println("CatalogController.handlePlaceOrder() - beginning the
transaction");
        userTransaction.begin();

        // 3. Insert the order into the database
        System.out.println("CatalogController.handlePlaceOrder() - inserting the
order in the database");
        OrderDB orderDB   new OrderDB();
        orderDB.insertOrder(order);

        // 4. Make the Web service call to CICS
        System.out.println("CatalogController.handlePlaceOrder() - calling the
CICS web service");
        orderDetails.populateResponse(
getOrderProxy(request).DFHZXCMNOperation(orderDetails.getRequestProgramIdentifier
()) );
        System.out.println("CatalogController.handlePlaceOrder() - response back
from the CICS web service");
        // 5. Throw exception if user ID ROLLBACK
        if(order.getUserId().equalsIgnoreCase("ROLLBACK"))
        {
           System.out.println("CatalogController.handlePlaceOrder() - simulating
the RemoteException");
           throw new RemoteException("Throwing the RemoteException");
        }
        // 6. Commit the transaction
        System.out.println("CatalogController.handlePlaceOrder() - commit the
transaction");
        userTransaction.commit();
        System.out.println("CatalogController.handlePlaceOrder()   after
commit");
     }
        // 7. Rollback in case of exception
     catch (Exception e)
     {
        try {
           System.out.println("CatalogController.handlePlaceOrder() -
exception thrown");
           userTransaction.rollback();
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000478

```
      } catch (Exception e1) {
         System.out.println("CatalogController.handlePlaceOrder() - Exception
when rollingback the transaction");
         e1.printStackTrace();
      }

      if (e.getMessage().startsWith("java.net.ConnectException"))
      {
         String errorMessage = "Unable to connect to service endpoint: "+
                        getOrderProxy(request).getEndpoint()+
                        "<BR> Please ensure service is running.";
         handleError(request, response, errorMessage);
         return;
      }
      else
      {
         e.printStackTrace();
         handleError(request, response, "An Error occured calling the service:
"+e.getMessage());
         return;
      }
   }
   // Call the response page setting the OrderDetails bean on the request
   RequestDispatcher dispatcher =
request.getRequestDispatcher(orderResponsePage);
   request.setAttribute("orderDetails",orderDetails);
   dispatcher.forward(request,response);
}
```

Note: Since the service requester application is a Web application (as opposed to an enterprise bean) we used a bean-managed transaction. For an enterprise bean, it is generally recommended to use container-managed transactions.

Change the deployment descriptor

The deployment descriptor of a J2EE application is used to specify whether the application component, if it makes any Web service requests, expects any transaction context to be propagated with the Web service requests (in accordance with the WebSphere WS-AT support).

To activate the WS-AT support:

- We imported the client application archive CatalogAtomic.ear into RAD. We then expanded the **Dynamic Web Project** (CatalogAtomicWeb) in the Project Explorer and opened (double-clicked) the **Deployment Descriptor** (Figure 12-4).

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 479/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000479

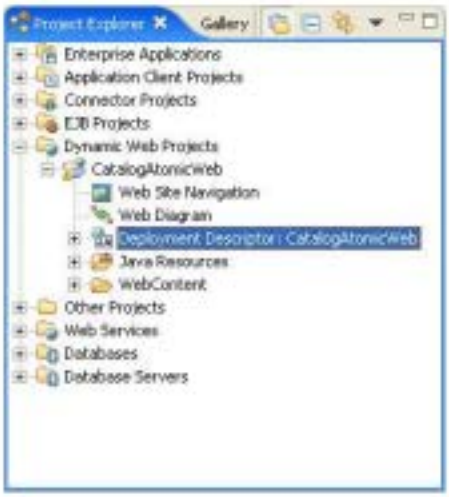


Figure 12-4 Project Explorer for CatalogAtomicWeb application

- In the Deployment Descriptor we clicked the **Servlets** tab and selected the **CatalogController** servlet. Then we scrolled down to Global Transaction and selected **Send Web Services Atomic Transactions on requests** (Figure 12-5).

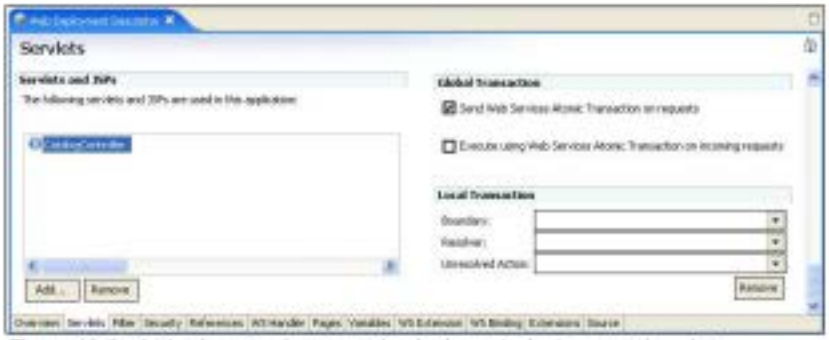


Figure 12-5 Activating atomic transaction in the web deployment descriptor

- We saved and closed the file.

Note: A similar deployment descriptor attribute **Use Web Services Atomic Transaction** can be used for enterprise beans.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 480/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000480

Create the ITSO.ORDER table

We created the ITSO.ORDER table to record all the orders that the Web browser user sends to the Catalog application. Example 12-7 shows the script we used to create the database and the table.

Example 12-7 Creation of ITSOWS database and ITSO.ORDER table

```
   CONNECT RESET;
-- Create database ITSOWS and schema ITSO
CREATE DATABASE ITSOWS;
CONNECT TO ITSOWS;
CREATE SCHEMA ITSO;

-- Table definitions for Order
CREATE TABLE ITSO.ORDER
  (ORDER_TMSTMP TIMESTAMP  NOT NULL ,
   ITEM_REF INTEGER  NOT NULL ,
   QUANTITY INTEGER  NOT NULL ,
   USER_ID CHARACTER (20)  NOT NULL ,
   CHARGE_DEPT CHARACTER (20)  NOT NULL  ,
   CONSTRAINT ORDERKEY PRIMARY KEY ( ORDER_TMSTMP)  ) ;
```

The ITSO.ORDER table has a column for every field in the Catalog Place Order JSP. Also, there is a TimeStamp column that is used as a unique key for the table. Figure 12-6 shows the ITSOWS database in the Control Center.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000481

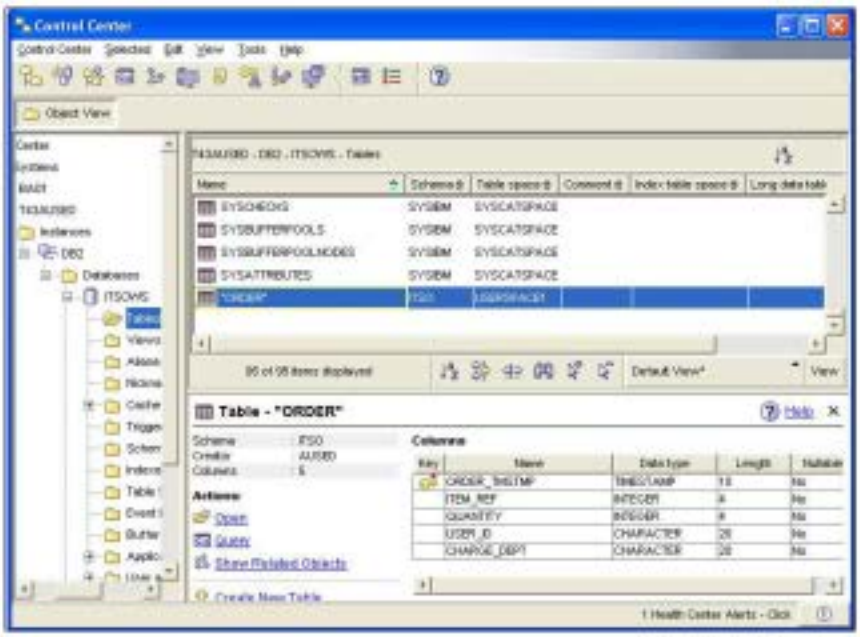


Figure 12-6 The ITSOWS database in the Control Center

Create a WebSphere data source

Next we configured the data source in WebSphere through the Administrative Console.

- We opened the Admin console by pointing a Web browser at:

  `http://cam21-pc11:9060/admin`

- In the main window, we opened **Resources** and then clicked **JDBC Providers** (Figure 12-7).

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 482/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000482

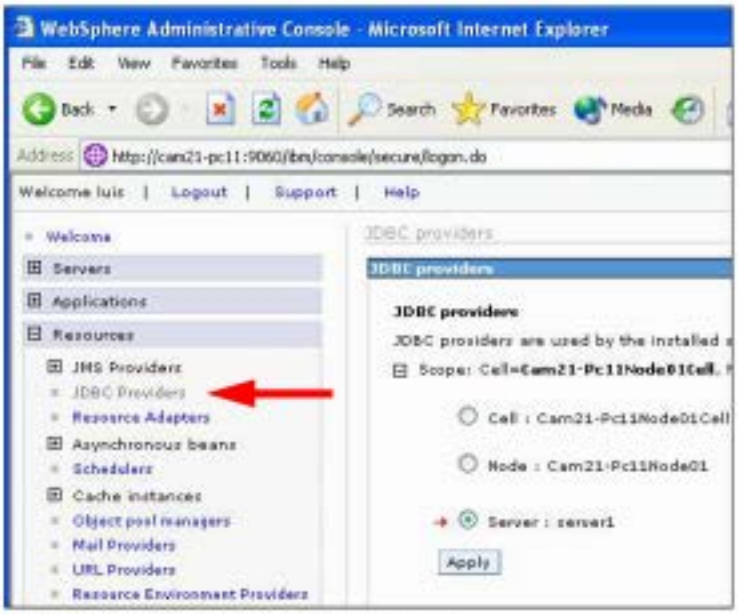


Figure 12-7 Admin console - JDBC providers

- Because we are running a single server, we chose the Server scope.
- To create a new JDBC provider we clicked **New**.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 483/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000483

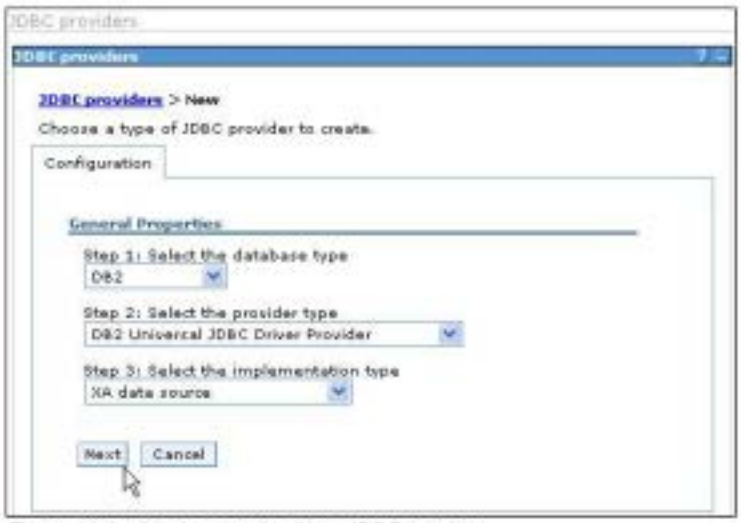


Figure 12-8 Admin console - New JDBC provider

- In the new JDBC provider window (Figure 12-8), we selected:
  - DB2 as database type
  - DB2 Universal JDBC Driver Provider as provider type
  - XA data source as the implementation type

  We clicked **Next**, and we were then presented with the window shown in Figure 12-9.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 484/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000484



Figure 12-9 Admin console - New JDBC provider 2

- We provided a name for the the JDBC provider, in this instance:

  `ITSO WebServices DB2 JDBC Driver Provider (XA)`

- We noted that WebSphere offers the Implementation class name COM.ibm.db2.jdbc.DB2XADataSource. This is the class of the DB2 driver that has two-phase commit capability. Since this class is not in the default

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 485/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000485

class path provided by WebSphere, we added to the end of the class path the following:

```
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2java.zip
```

- We clicked **OK** and saved the configuration changes.

Now we have the new JDBC provider defined and we can see it in the list of JDBC providers.

Define the DB2UNIVERSAL_JDBC_DRIVER_PATH variable

We defined the DB2UNIVERSAL_JDBC_DRIVER_PATH WebSphere variable as follows:

- In the Admin console main window we clicked **Environment** and then clicked **WebSphere Variables** as shown in Figure 12-10.



Figure 12-10 Admin console - WebSphere variables window

- We set the value of the DB2UNIVERSAL_JDBC_DRIVER_PATH variable to the default JDBC driver location C:\Program Files\IBM\SQLLIB\java (Figure 12-11).

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 486/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000486



Figure 12-11 Admin console - Defining the DB2 driver path variable

Create the J2C Authentication data

Before creating the data source, we needed to create the J2EE Connector Architecture (J2C) Authentication data in JAAS Configuration.

- In the Admin console main window we selected **Security** and then **Global Security**.
- In the Global security page we clicked **JAAS Configuration under Authentication**, and selected **J2C Authentication data**.
- In the **J2EE Connector Architecture (J2C) authentication data entries** page we clicked **New**. The window shown in Figure 12-12 was displayed.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000487



Figure 12-12 Admin console - Define J2C authentication alias

- In the General Properties we entered the following values:
  - DB2user as Alias
  - Administrator as User ID
  - User password as Password
- We clicked **OK** and saved the configuration changes.

Configure the data source for the JDBC provider

Next we configured the data source for the JDBC provider.

- In the Admin console main window we selected **Resources** and then **JDBC Providers**.
- In the JDBC providers window, we clicked the new provider that we defined in "Create a WebSphere data source" on page 453:

  `ITSO WebServices DB2 JDBC Driver Provider (XA)`

- In **Additional Properties** we clicked **Data sources**, then clicked **New**. We were presented with the window shown in Figure 12-13.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
Online/Notary.net
Page 488/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000488

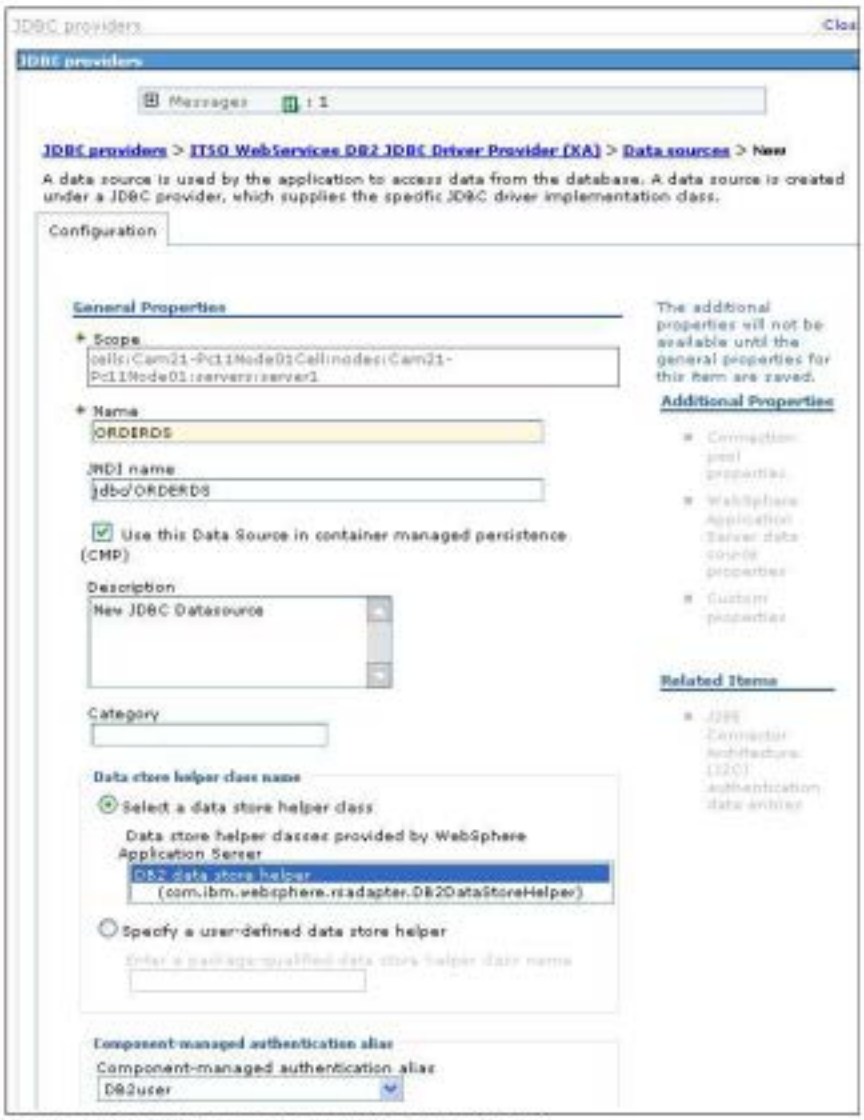


Figure 12-13 Admin console - New Data source (1 of 2)

- In the General Properties we entered the following values:
  - ORDERDS as the Data source name.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 489/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000489

- jdbc/ORDERDS as the JNDI name.
- DB2user as the Component-managed authentication alias.

- We then scrolled down to the bottom of the window and entered ITSOWS as the Database name (Figure 12-14).

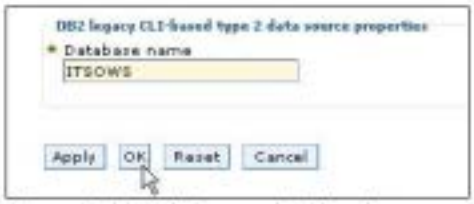


Figure 12-14 Admin console - New data source (2 of 2)

- We clicked **OK** and then saved the changes.

12.2.3 Testing the simple scenario

We performed two tests of the simple scenario:

- Normal transaction termination
- Abnormal transaction termination (see "Simple scenario: abnormal transaction termination" on page 479)

Simple scenario: normal transaction termination

In this section, we explain how we ran the AtomicClient application and then show the registration and protocol service messages that are exchanged between WebSphere Application Server and CICS during the normal termination of the atomic transaction.

To run the scenario we followed these steps:

- We opened the welcome window of the Catalog application using the URL:

  `http://cam21-pc11:9080/CatalogAtomicWeb/Welcome.jsp`

  We were presented with the window shown in Figure 12-15.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000490

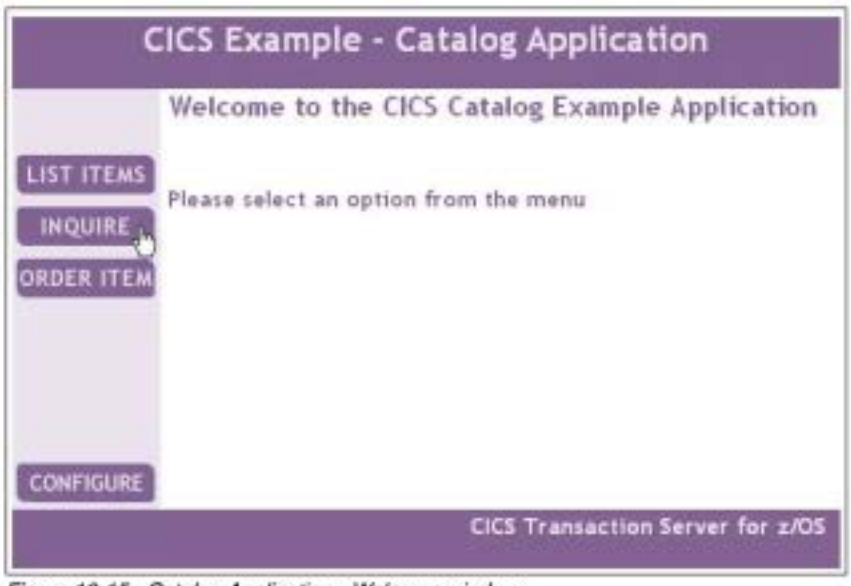


Figure 12-15 Catalog Application - Welcome window

- We clicked **INQUIRE**.
- In the Inquire Single window we used the Item Reference Number default value of 0010 and clicked **SUBMIT**. The Web service request was sent to CICS and we were presented with the results of the inquiry as shown in Figure 12-16.



Figure 12-16 Catalog Application - Inquire single

- We noted that the number of items in stock was 76. This value is taken from the CICS VSAM file.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 491/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000491

- We clicked **SUBMIT** to go to the Enter Order Details window shown in Figure 12-17.



Figure 12-17 Catalog Application - Enter order details

- In the Enter Order Details window we provided a User Name and a Department Name and clicked **SUBMIT**.
- After the CICS Web service processed the order, we got a response telling us that the order was successfully placed (Figure 12-18).



Figure 12-18 Catalog Application - Order placed response

In the WebSphere server log we see trace entries, generated by the Catalog Controller servlet, which show that the transaction was successfully committed (Example 12-8).

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 492/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000492

Example 12-8 WebSphere server log with successful Place Order

```
CatalogContoller:doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - commit the transaction
CatalogController.handlePlaceOrder() - after commit
```

- Next we checked the same item number through the Inquire Single service and verified that the stock level decreased by one item (Figure 12-19).

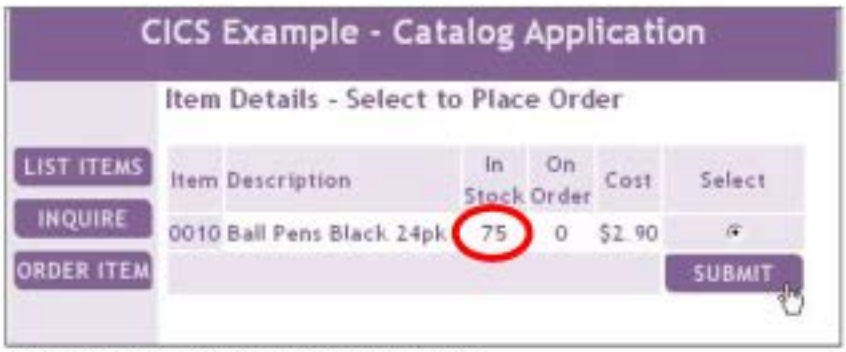


Figure 12-19 Catalog Application - Inquire single

- We opened a DB2 Control Center and issued the SQL command:

  ```
  SELECT * FROM ITSO.ORDER
  ```

  This lists all of the records in our ITSO.ORDER table. Figure 12-20 shows that our new record is in the table.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000493

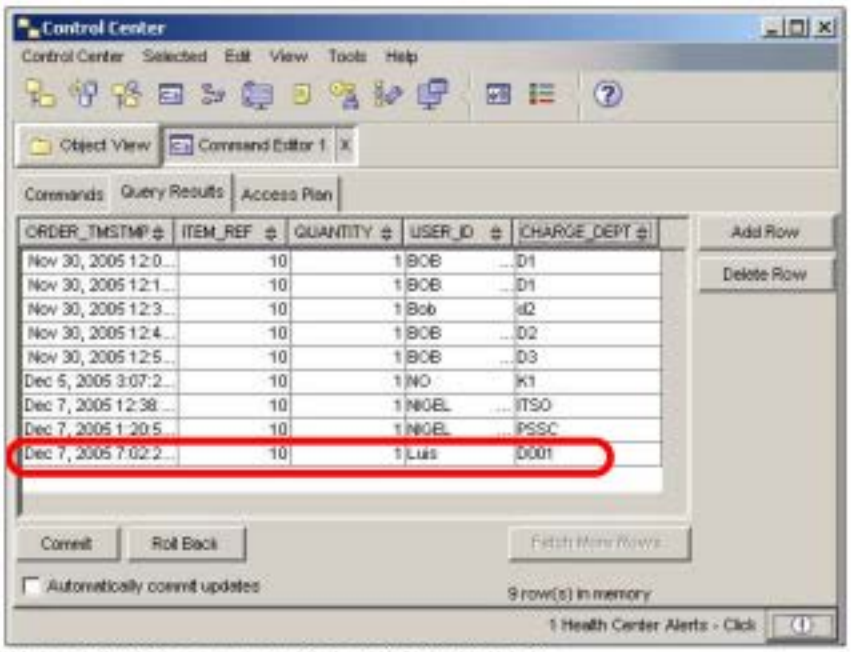


Figure 12-20 The new record in the ITSO .ORDER table

Figure 12-21 shows the registration and protocol service messages that were exchanged between CICS and WebSphere Application Server during our test. Note that WebSphere Application Server uses separate endpoint addresses for its Registration service, Protocol service, and fault messages.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 494/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000494

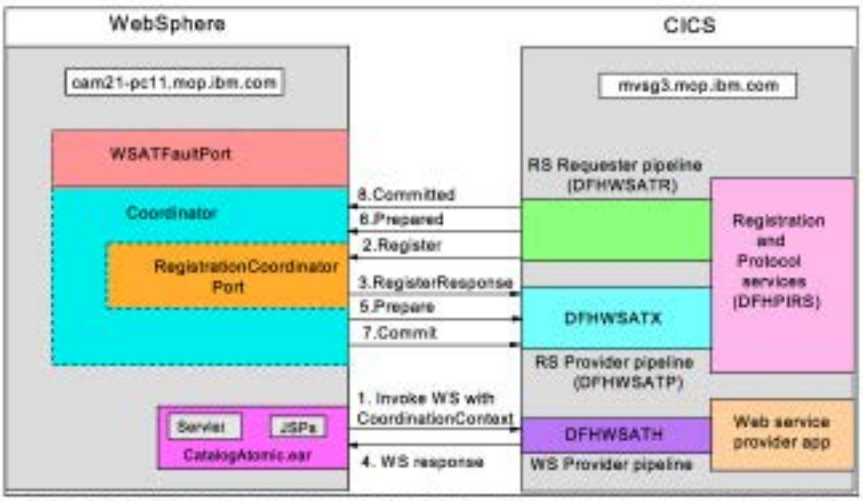


Figure 12-21 Messages exchanged during test of normal atomic transaction termination

We conclude our discussion of this test by showing the complete messages summarized by Figure 12-21. To make each message easier to understand we:

- Format it
- Replace the URL in each xmlns attribute with "..." for each namespace in Example 12-9. For instance, we replaced xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" with xmlns:soap="..." .

Example 12-9 Namespaces used in messages between WebSphere and CICS

```
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wscoor="http://schemas.xmlsoap.org/ws/2004/10/wscoor"
xmlns:wsat="http://schemas.xmlsoap.org/ws/2004/10/wsat"
xmlns:websphere-wsat="http://wstx.Transaction.ws.ibm.com/extension"
xmlns:cicswsat="http://www.ibm.com/xmlns/prod/CICS/pipeline"
```


Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 485/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000495

Invoke Web service with coordination context (message 1)

Example 12-10 shows the SOAP 1.1 message that WebSphere sends to CICS to invoke the placeOrder Web service. The message consists of a SOAP envelope that contains a SOAP header and a SOAP body.

Example 12-10 WebSphere sends to CICS a Web service request with a CoordinationContext header

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
xmlns:xsi="..."
                  xmlns:wscoor="..." xmlns:wsa="...">
   <soapenv:Header>
      <wscoor:CoordinationContext soapen:mustUnderstand="1">
         <wscoor:Expires>never</wscoor:Expires>
         <wscoor:Identifier>
            com.ibm.ws.wstx:0000010810a19b020000000100000006b3e8b0
            ecc989b8eab92e29d91254fd41b0f147b9
         </wscoor:Identifier>
         <wscoor:CoordinationType>
            http://schemas.xmlsoap.org/ws/2004/10/wsat
         </wscoor:CoordinationType>
         <wscoor:RegistrationService xmlns:wscoor="...">
            <wsa:Address xmlns:wsa="...">

http://9.100.199.238:9080/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
            </wsa:Address>
            <wsa:ReferenceProperties xmlns:wsa="...">
               <wsphere-wsat:txID xmlns:wsphere-wsat="...">
                  com.ibm.ws.wstx:0000010810a19b020000000100000006b3e8b0
                  ecc989b8eab92e29d91254fd41b0f147b9
               </wsphere-wsat:txID>
               <wsphere-wsat:instanceID xmlns:wsphere-wsat="...">
                  com.ibm.ws.wstx:0000010810a19b020000000100000006b3e8b0
                  ecc989b8eab92e29d91254fd41b0f147b9
               </wsphere-wsat:instanceID>
            </wsa:ReferenceProperties>
         </wscoor:RegistrationService>
      </wscoor:CoordinationContext>
   </soapenv:Header>
   <soapenv:Body>
      <p635:DFHCICMN xmlns:p635="http://www.DFHCICMN.DFHCICP5.Request.com">
         <p635:ca_request_id>CICSCI</p635:ca_request_id>
         <p635:ca_return_code>0</p635:ca_return_code>
         <p635:ca_response_message></p635:ca_response_message>
         <p635:ca_order_request>
            <p635:ca_userid>Luis</p635:ca_userid>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000496

```
            <p635:ca_charge_dept>DOOI</p635:ca_charge_dept>
            <p635:ca_item_ref_number>10</p635:ca_item_ref_number>
            <p635:ca_quantity_req>1</p635:ca_quantity_req>
            <p635:filler1 xsi:nil="true"/>
         </p635:ca_order_request>
      </p635:DFHOXCMN>
   </soapenv:Body>
</soapenv:Envelope>
```

- SOAP header

  The SOAP header contains only one header, a `CoordinationContext` header. The `CoordinationContext` header has a `mustUnderstand` attribute whose value is `1`. This means that CICS must process the `CoordinationContext` header. If CICS cannot process the `CoordinationContext` header (for example, because the provider pipeline configuration file does not specify the DFHWSATH message handler) then CICS must stop all further processing of the message and generate a SOAP fault whose fault code is `MustUnderstand`.

  The `CoordinationContext` element contains four elements:

  - `Expires`

    CICS treats the content of the `Expires` element as a character string and determines whether the string represents a number.

    - If it does, it is treated as a millisecond value, which is then converted to an integer representing the number of seconds for which the Web service transaction waits for a response to a **`Register`** request or for the Coordinator to send various 2PC messages. If this integer is 0, or greater than 4080, then CICS uses 4080 seconds (the maximum value which CICS allows for the DTIMOUT attribute of the TRANSACTION resource definition).
    - If it does not, then CICS does not set a suspend time and the Web service transaction waits forever.

    Note: We noted that the Expires element was not an unsigned integer but the text `Never`. The default CICS behavior is not to terminate the transaction but to wait indefinitely for WebSphere to commit or roll back the atomic transaction.

  - `Identifier`

    WebSphere creates this identifier as a unique global identifier for each WS-AT transaction as required by the WS-Coordination specification.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000497

CoordinationType

The CoordinationType element specifies the WS-AtomicTransaction coordination type.

- RegistrationService

  The RegistrationService element contains an endpointReference for WebSphere's Registration service. The endpointReference has two elements:

  - Address

    The Address element tells CICS where to send its **Register** request. CICS copies the contents of the Address element to the To message information header when it builds its **Register** request.

  - ReferenceProperties

    WebSphere provides two reference properties: txID and instanceID. The content of the txID element is the same as the content of the Identifier element. The content of the instanceID element is initially the same as that of the txID element, but that will change later.

    CICS adds these reference properties to the SOAP header when it builds its **Register** request. This will allow WebSphere to map the **Register** request to this service request. The detail which WebSphere puts into the txID and instanceID properties is implementation specific and is only ever parsed and understood by WebSphere.

- SOAP body

  The content of the <p635:ca_request_id> element tells CICS which Web service to invoke; 0'ORDR indicates the placeOrder Web service. The contents of the <p635:ca_userid>, <p635:ca_charge_dept>, <p635:ca_item_ref_number>, and <p635:ca_quantity_req> elements provide the placeOrder Web service with the details of the order (see Figure 12-17 on page 463).

Register (message 2)

Example 12-11 shows the SOAP message which contains the **Register** request that CICS sends to WebSphere. The message consists of a SOAP envelope which contains a SOAP header and a SOAP body.

Example 12-11 CICS sends a Register request to WebSphere

```
<soap:Envelope xmlns:wscoor="..." xmlns:wsa="..." xmlns:cicswsat="..."
xmlns:soap="...">
   <soap:Header>
      <wsa:Action>
         http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register
```


Document Id: [illegible] Page 469(of)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000498

```
      </wsa:Action>
      <wsa:MessageID>FIAT MSG A62E73C1 3033431450526270</wsa:MessageID>
      <wsa:ReplyTo>
         <wsa:Address>
            http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
         </wsa:Address>
         <wsa:ReferenceProperties>
            <cicswsat:UOWID>BEC92025168B596B</cicswsat:UOWID>
            <cicswsat:PublicId>
               310FD7E2E2C3C7F34BC1F6D7D6E3F3C3F10FD7E2E2C3C7F34BC1F6D7D6E3F3C3
               F1C3C9F6F2F3C44040AF09202616780000092029167B0C0C4040404040404040
            </cicswsat:PublicId>
         </wsa:ReferenceProperties>
      </wsa:ReplyTo>
      <wsa:To>

http://9.100.199.238:9080/ IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
      </wsa:To>
      <websphere-wsat:txID xmlns:websphere-wsat="...">
         com.ibm.ws.wstx:3030313810a19b020000000100000006d3e8b0
         ecc899b8eab92e29d91254fc4fb0ff47b9
      </websphere-wsat:txID>
      <websphere-wsat:instanceID xmlns:websphere-wsat="...">
         com.ibm.ws.wstx:3030313810a19b020000000100000006d3e8b0
         ecc899b8eab92e29d91254fc4fb0ff47b9
      </websphere-wsat:instanceID>
   </soap:Header>
   <soap:Body>
      <wscoor:Register>
         <wscoor:ProtocolIdentifier>
            http://schemas.xmlsoap.org/ws/2004/10/wsat/Durable2PC
         </wscoor:ProtocolIdentifier>
         <wscoor:ParticipantProtocolService>
            <wsa:Address>
               http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
            </wsa:Address>
            <wsa:ReferenceProperties>
               <cicswsat:UOWID>BEC92025168B596B</cicswsat:UOWID>
               <cicswsat:PublicId>
                  310FD7E2E2C3C7F34BC1F6D7D6E3F3C3F10FD7E2E2C3C7F34BC1F6D7D6E3F3C3
                  F1C3C9F6F2F3C44040AF09202616780000092029167B0C0C4040404040404040
               </cicswsat:PublicId>
            </wsa:ReferenceProperties>
         </wscoor:ParticipantProtocolService>
      </wscoor:Register>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000499

```
</soap:Body>
</soap:Envelope>
```

- SOAP header

  The SOAP header contains several message information headers and the two reference properties (txID and instanceID) which WebSphere sent in its Web service request. These reference properties allow WebSphere to match the **Register** request to its initial request for the placeOrder service. The message information headers are as follows.

  - To

    The To header shows that the message is being sent to the Registration Coordinator Port running in a WebSphere Application Server V6.0 region which is monitoring port 9080 on a system whose IP address is 9.100.199.238. CICS copied this from the Address element of the endpointReference for the RegistrationService in WebSphere's service request.

  - Action

    The Action header indicates that CICS wishes to register with the Registration Coordinator Port.

  - MessageID

    The ID of the message uniquely identifies the message in space and time:

    - PIAT is part of the name of module DFHPIAT. DFHPIAT generates CoordinationContext elements for CICS and interprets CoordinationContext elements received from other systems.
    - A6P0T3C1 is the VTAM APPLID of the CICS region which is sending the message.
    - [illegible] is the abstime value returned by an EXEC CICS INQUIRE TIME issued in our CICS region.

  - ReplyTo

    The Address element of the ReplyTo header shows that the reply to this message should be sent to the Registration service running in a CICS region which is monitoring port 15301 on a z/OS system whose IP address is MVSG3.mop.ibm.com. This Registration service has two reference properties:

    - UOWID
    - PublicID

    When WebSphere sends the **RegisterResponse** to CICS, it will add each of these reference properties to the response as a SOAP header. CICS

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000500

will use the PublicID to find the region and the UOW in that region where the Web service provider is waiting for the response; then CICS will route it there for processing. The detail which CICS puts into the UOWID and PublicID properties is there for CICS to understand; other products do not use it.

- SOAP body

  The SOAP body contains the **Register** request, which contains two elements:

  ProtocolIdentifier

  CICS uses this element to register for the durable two-phase commit protocol

  - ParticipantProtocolService

    Note that the address of CICS's Protocol service is the same as the address of its Registration service.

RegisterResponse (message 3)

Example 12-12 shows the **RegisterResponse** that WebSphere sends to CICS.

Example 12-12 WebSphere sends a RegisterResponse to CICS

```
<soapenv:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
xmlns:xsi="..."
                  xmlns:wsa="...">
   <soapenv:Header>
      <wsa:MessageID>uuid:1CA19035-0108-4000-E000-094403D46363</wsa:MessageID>
      <wsa:To>http://MVSG3.mop.ibm.com:15301/c:cswsal/RegistrationService</wsa:To>
      <wsa:Action>
         http://schemas.xmlsoap.org/ws/2004/10/wscoor/RegisterResponse
      </wsa:Action>
      <wsa:FaultTo xmlns:wsa="...">
         <wsa:Address>
            http://9.103.199.230:9080/ IBMSYSAPP/wsalfault/services/WSATFault?wsdl
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere-wsat:txID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:00000106101a19b020000000100000068Je6b0
               ec:899b8ea92e29d91254fc4fb0ff47a9
            </websphere-wsat:txID>
            <websphere-wsat:instanceID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:00000106101a19b020000000100000068Je6b0
               ec:899b8ea92e29d91254fc4fb0ff47a9
            </websphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:FaultTo>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000501

```
      <wsa:From xmlns:wsa="...">
         <wsa:Address>

http://9.100.199.238:9080/_IBMSYSAPP/wscoor/services/RegistrationCoordinatorPort
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere wsat:txID xmlns:websphere wsat="...">
               com.ibm.ws.wsat:0000010810a19b320D3D3D310D0D0D0E83e8b0
               ecc889b8eab92e29d91254fc4fb01f47b9
            </websphere-wsat:txID>
            <wshsphere-wsat:instanceID xmlns:wshsphere-wsat="...">
               com.ibm.ws.wsat:0000010810a19b320D0D0D0D10D0D0D0E83e8b0
               ecc889b8eab92e29d91254fc4fb01f47b9
            </websphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:From>
      <wsa:RelatesTo>F1AT MSG A6P3T3C1 3033431450526270</wsa:RelatesTo>
      <cicswsat:UOWID xmlns:cicswsat="..."><E3920Z5168B5956</cicswsat:UOWID>
      <cicswsat:PublicId xmlns:cicswsat="...">
         31CFD7E2E2C3C7F348C1F607D6E3F3C3F1C0F07E2E2C3C7F34BC1F6070E3F3C3
         F1C3C9F6F2F3C44040F0970261678000009702616780000404040404040404040
      </cicswsat:PublicId>
   </soapenv:Header>
   <soapenv:Body>
      <RegisterResponse xmlns="http://schemas.xmlsoap.org/ws/2004/10/wscoor">
         <wscoor:CoordinatorProtocol$ervice xmlns:wscoor="...">
            <wsa:Address xmlns:wsa="...">
               http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator
            </wsa:Address>
            <wsa:Reference Properties xmlns:wsa="...">
               <websphere-wsat:txID xmlns:websphere-wsat="...">
                  com.ibm.ws.wsta:0000010810a19b320D0D0D0D10D0D0D0E83e8b0
                  ecc889b8eab92e29d91254fd41b3f47b9
               </websphere-wsat:txID>
               <websphere-wsat-instanceID xmlns:websphere-wsat="...">
                  http://W5G3.mop.ibm.com:16301/cicswsat/RegistrationService
               </websphere wsat:instanceID>
            </wsa:ReferenceProperties>
         </wscoor:CoordinatorProtocol Service>
      </RegisterResponse>
   </soapenv:Body>
</soapenv:Envelope>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000502

- SOAP header

  The SOAP header contains several message information headers and the two reference properties (UOW[]) and Public[]) which CICS sent in its **Register** request. If we were running a CICSPlex, CICS would use these reference properties to route the response to the region and the UOW in that region where the Web service provider is waiting for it. The message information headers are as follows:

  - MessageID

    To

    WebSphere is sending this message to CICS's RegistrationService. It obtained this address from the ReplyTo header in the **Register** request.

  - Action

    This message is a **RegisterResponse**.

    FaultTo

    WebSphere wants CICS to send any SOAP faults that it has to generate to WebSphere's WSATFaultPort.

  - From

    This message is coming from WebSphere's RegistrationCoordinatorPort.

    RelatesTo

    This message relates to the **Register** request which CICS sent.

- SOAP body

  The SOAP body contains the RegisterResponse element, which in turn contains only a CoordinatorProtocolService element. The address of WebSphere's Protocol Service is http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator. Note that its two reference properties, txID and instanceID, no longer have the same contents, instanceID now contains the address of CICS's Registration service.

Web service response (message 4)

Since CICS has registered its interest in the atomic transaction and WebSphere has responded to that, CICS can now run the placeOrder service. When the placeOrder service has completed its work, CICS sends a response to WebSphere as shown in Example 12-13. The essence of the response is the message ORDER SUCCESSFULLY PLACED.

Example 12-13 CICS sends the Web service response to WebSphere

```
<SOAP-ENV:Envelope xmlns:soapenv="..." xmlns:soapenc="..." xmlns:xsd="..."
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000503

```
                xmlns:xsi="..."     xmlns:wscoor="..."  xmlns:wsa="..."
                xmlns:SOAP-ENV="...">
   <SOAP-ENV:Body>
      <DFHOXCMNResponse xmlns="http://www.DFHOXCMN.DFHOXCP5.Response.com">
         <ca_request_id>01ORDR</ca_request_id>
         <ca_return_code>0</ca_return_code>
         <ca_response_message>ORDER SUCESSFULLY PLACED</ca_response_message>
         <ca_order_request>
            <ca_userid>Luis     </ca_userid>
            <ca_charge_dept>0001    </ca_charge_dept>
            <ca_item_ref_number>10</ca_item_ref_number>
            <ca_quantity_req>1</ca_quantity_req>
            <filler>
            ...
            </filler>
         </ca_order_request>
      </DFHOXCMNResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Prepare (message 5)

When the handlePlaceOrder method of the CatalogController servlet issues the **userTransaction.commit** command, WebSphere sends a **Prepare** command to CICS as shown in Example 12-14 in the Action message information header and in the <p320:Prepare.../> element of the SOAP Body.

Example 12-14 WebSphere sends a Prepare notification to CICS

```
<soapenv:Envelope   xmlns:soapenv="..."   xmlns:soapenc="..."   xmlns:xsd="..."
                    xmlns:xsi="..."       xmlns:wsa="...">
   <soapenv:Header>
      <wsa:MessageID>uuid:1DA19D3B 3D36 4DD1 E000 0944D9D4B3B9</wsa:MessageID>
      <wsa:To>http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationServices</wsa:To>
      <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepare</wsa:Action>
      <wsa:FaultTo xmlns:wsa="...">
         <wsa:Address>
            http://9.100.199.230:3080/ IBMSYSAPP/wsatfault/services/WSATFault?wsdl
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere-wsat:txID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:000001810a19b070000000010000000683e6b0
               ec:c809b0ea:1a92e29:191254fc:47b0ff47a9
            </websphere-wsat:txID>
            <websphere-wsat:instanceID xmlns:websphere-wsat="...">
               http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000504

```
            </wsbsphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:FaultTo>
      <wsa:ReplyTo xmlns:wsa="...">
         <wsa:Address>
            http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere-wsat:txID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:30303130610a19b020000030010000000083e8b0
               ecc899h8eah92c29d91254fc4fh0ff47a9
            </websphere-wsat:txID>
            <websphere-wsat:instanceID xmlns:websphere-wsat="...">
               http://MVS33.nap.ibm.com:1530L/cicswsat/RegistrationService
            </websphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:ReplyTo>
      <cicswsat:UOWID xmlns:cicswsat="...">BE39202516dB5956</cicswsat:UOWID>
      <cicswsat:PublicId xmlns:cicswsat="...">
         31CFD7E2E2C3C7F340C1F607D6E3F3C3F1C0D7E2E2C3C7F34BC1FC070CE3F3C3
         F1C3C9F6F2F3C44040AF39702516760000097025167B0000404040404040404040
      </cicswsat:PublicId>
   </soapenv:Header>
   <soapenv:Body>
      <p320:Prepare xsi:nil="true"
xmlns:p320="http://schemas.xmlsoap.org/ws/2004/10/wsat"/>
   </soapenv:Body>
</soapenv:Envelope>
```

Prepared (message 6)

CICS responds with **Prepared** as shown in Example 12-15 in the Action message information header and in the <wsat:Prepared.../> element of the SOAP Body. Since **Prepared** is a terminating message, it does not contain a ReplyTo message information header.

Example 12-15 CICS sends Prepared notification to WebSphere

```
<soap:Envelope xmlns:xscoor="..." xmlns:wsa="..." xmlns:cicswsat="..."
xmlns:soap="...">
   <soap:Header>
      <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepared</wsa:Action>
      <wsa:MessageID>FEAT MSG A6PST3C1 30334314505327DC</wsa:MessageID>
      <wsa:To>http://9.100.199.238:9080/_IBMSYSAPP/wsat/services/Coordinator</wsa:To>
      <websphere-wsat:txID xmlns:websphere-wsat="...">
         com.ibm.ws.wstx:30303130610a19b020000030010000000083e8b0
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000505

```
            ecc889b8eab92e29d91254fc4fb0ff47b9
        </websphere-wsat:txID>
        <websphere-wsat:instanceID xmlns:websphere-wsat="...">
            http://MVS03.mop.ibm.com:15301/cicswsat/RegistrationService
        </websphere-wsat:instanceID>
    </soap:Header>
    <soap:Body>
        <wsat:Prepared xmlns:wsat="..."/>
    </soap:Body>
</soap:Envelope>
```

Commit (message 7)

Since CICS has voted **yes** in response to the **Prepare** command, and since no other system has registered an interest in this transaction, WebSphere sends a **Commit** command to CICS. See the Action header and the <p320:Commit.../> element of the SOAP Body in Example 12-16

Example 12-16 WebSphere sends Commit notification to CICS

```
<soapenv:Envelope xmlns:soapenv="..."  xmlns:soapenc="..." xmlns:xsd="..."
xmlns:xsi="..."
                  xmlns:wsa="...">
   <soapenv:Header>
      <wsa:MessageID>uuid:10A19F67-1C38-4000-E000-09F409D4B353</wsa:MessageID>
      <wsa:To>http://MVS03.mop.ibm.com:15301/cicswsat/RegistrationService</wsa:To>
      <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Commit</wsa:Action>

      <wsa:FaultTo xmlns:wsa="...">
         <wsa:Address>
            http://9.100.199.238:9080/_IBMSYS4PP/wsatfault/services/WSATFaultPort
         </wsa:Address>
         <wsa:ReferenceProperties xmlns:wsa="...">
            <websphere-wsat:txID xmlns:websphere-wsat="...">
               com.ibm.ws.wstx:00000C0810a19b020000000100000000683eBb0
               ecc889b8eab92e29d91254fc4fb0ff47b9
            </websphere-wsat:txID>
            <websphere-wsat:instanceID xmlns:websphere-wsat="...">
               http://MVS03.mop.ibm.com:15301/cicswsat/RegistrationService
            </websphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:FaultTo>
      <wsa:ReplyTo xmlns:wsa="...">
         <wsa:Address>
            http://9.100.199.238:9080/_IBMSYS4PP/wsat/services/Coordinator
         </wsa:Address>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000506

```
         <wsa:ReferenceProperties xmlns:wsa="...">
            <webSphere wsat:txID xmlns:webSphere wsat="...">
               com.ibm.ws.wstx:00000130810a19b320000000010000000E83e8b0
               ecc889b8eab92e29d91254fc4fb0ff47b9
            </webSphere-wsat:txID>
            <webSphere-wsat:instanceID xmlns:webSphere-wsat="...">
               http://MVS53.mop.ibm.com:1530I/cicswsat/RegistrationService
            </webSphere-wsat:instanceID>
         </wsa:ReferenceProperties>
      </wsa:ReplyTo>
      <cicswsat:UOWID xmlns:cicswsat="...">9E092026168B566B</cicswsat:UOWID>
      <cicswsat:PublicId xmlns:cicswsat="...">
         31CFD7E2E2C3C7F34BC1F60706E3F3C3F1CF37E2E2C3C7F34BC1FED7DEE3F3C3
         F1C3C9E6E2F3C4404D8E3D9202516760D0D09202516760D0D0404040404040404040
      </cicswsat:PublicId>
   </soapenv:Header>
   <soapenv:Body>
      <p320:Commit xsi:nil="true"
xmlns:p320="http://schemas.xmlsoap.org/ws/2004/10/wsat"/>
   </soapenv:Body>
</soapenv:Envelope>
```

Committed (message 8)

After committing the update to the EXMPCAT VSAM file, CICS sends a **Committed** notification to WebSphere. See the Action message information header and the <wsat:Committed .../> element of the SOAP Body in Example 12-17. Since **Committed** is a terminating message, it does not contain a ReplyTo message information header.

Example 12-17 CICS sends Committed notification to WebSphere

```
<soap:Envelope   xmlns:wsaw="..."   xmlns:wsa="..."   xmlns:cicswsat="..."
                 xmlns:soap="...">
   <soap:Header>
      <wsa:Action>http://schemas.xmlsoap.org/ws/2004/10/wsat/Committed</wsa:Action>
      <wsa:MessageID>PT4T-H5G-A6P3T3C1-3033431460533240</wsa:MessageID>
      <wsa:To>tcp://9.100.199.230:9080/ IBMSYSAPP/wsat/services/Coordinator</wsa:To>
      <webSphere-wsat:txID xmlns:webSphere-wsat="...">
         com.ibm.ws.wstx:00000130810a19b320000000010000000E83e8b0
         ecc889b8eab92e29d91254fc4fb0ff47b9
      </webSphere-wsat:txID>
      <webSphere wsat:instanceID xmlns:webSphere wsat="...">
         http://MVS53.mop.ibm.com:1530I/cicswsat/RegistrationService
      </webSphere-wsat:instanceID>
   </soap:Header>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000507

```
   <soap:Body>
      <wsat:Committed xmlns:wsat="..."/>
   </soap:Body>
</soap:Envelope>
```

Let's think back to Waldo and his money transfer transaction (see "Beginner's guide to atomic transactions" on page 382). What Waldo requires is that both databases cooperate to ensure that they are always in a known and consistent state.

The set of eight Web service, registration, and protocol service messages we have described here are the flows that guarantee database consistency in our scenario, a scenario in which the databases are accessed by middleware (WebSphere Application Server on a Windows platform and CICS on z/OS), and communication between the middleware products is based on open Web services standards. The same basic flows could be used between any middleware products which support the Web services, Web Services-Coordination (WS-C) and Web Services-Atomic Transaction (WS-AT) specifications.

Simple scenario: abnormal transaction termination

We return to the Place Order window to place a new order. This time we used ROLLBACK as the User Name. This tells the servlet to throw an exception after inserting the order in the DB2 table and placing the order with CICS.

Example 12-18 CatalogController servlet: Throwing the RemoteException

```
if(order.getUserId().equalsIgnoreCase("ROLLBACK"))
{
   System.out.println("CatalogController.handlePlaceOrder() - Simulating the
RemoteException");
   throw new RemoteException("Throwing the RemoteException");
}
```

On the Order Entry page we entered the User Name ROLLBACK and clicked **SUBMIT** to place a new order (Figure 12-22). Note that, at this time, we still had 75 items in stock (see Figure 12-19 on page 464).

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000508



Figure 12-22 Catalog Application - Place order for ROLLBACK

Figure 12-23 shows the expected error response.

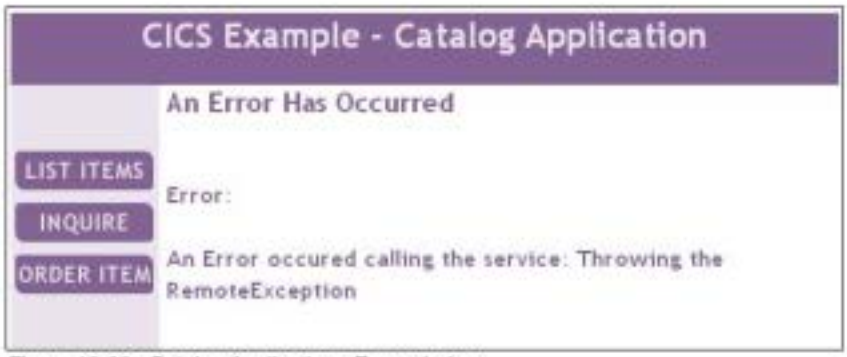


Figure 12-23 Catalog Application - Error window

When the Exception is thrown, the transaction is rolled back; the order is deleted from the table and the CICS transaction is rolled back too.

We again check in the WebSphere server log to see what happened (Example 12-19).

Example 12-19 WebSphere server log with ROLLBACK Place Order

```
CatalogContoller:doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
```


Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineHistory.net
Page 509/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000509

```
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - simulating the RemoteException
CatalogController.handlePlaceOrder() - rollingback the transaction
```

We can see in the log that the order was inserted in the database and a good response was returned from calling the CICS Web service. Then we simulated the RemoteException and the rollback of the whole transaction took place.

When we inquired on the stock level, we noted that there were still 75 items in stock.

Figure 12-24 shows the registration and protocol service messages that CICS and WebSphere Application Server exchanged during our test.

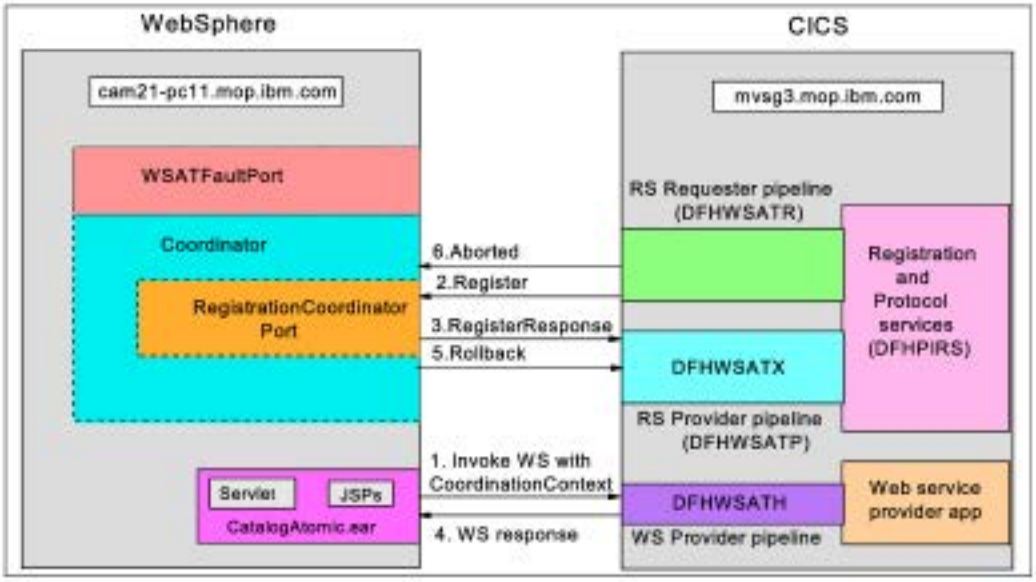


Figure 12-24 Messages exchanged when both sides back out

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 510/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000510

12.3 The daisy chain atomic transaction scenario

We now extend the simple atomic transaction scenario to a *daisy chain* scenario by making the following modifications:

- Changing the configuration of the CICS sample Catalog application such that it makes an outbound Web service call
- Changing the ExampleAppDispatchOrder.ear file provided with CICS TS V3.1

  ExampleAppDispatchOrder.ear is an enterprise archive provided with the Catalog application that can be deployed in WebSphere Application Server and used as an order dispatch endpoint. See "Catalog manager example application" on page 67 for more information on the sample application.

  We modified the ExampleAppDispatchOrder.ear file to insert records into a DB2 table called ITSO.DISPATCH. The ITSO.DISPATCH table logs all of the orders dispatched through the dispatchOrder Web service. We called the modified program DispatchOrderAtomic.ear.

By making these changes, we have a global transaction that updates three resources:

- A DB2 table in the Windows environment
- A VSAM file in the z/OS environment
- Another DB2 table in a Windows environment

We have a daisy chain from WebSphere to CICS to WebSphere. CICS acts as both a coordinator and a participant. Figure 12-25 shows the sequence of events for the whole transaction.

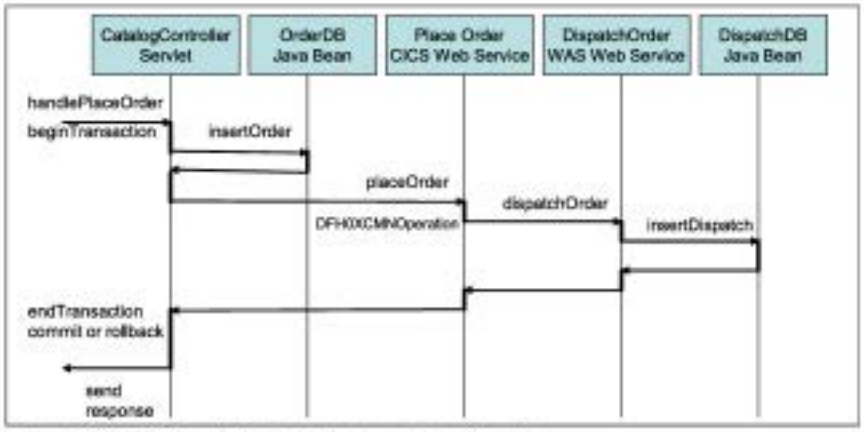


Figure 12-25 Daisy chain scenario sequence of events

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 511/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000511

Figure 12-26 shows a more global view of the sequence of events

- The user uses his Web browser to invoke the AtomicClient CatalogController servlet, which runs in WebSphere Application Server
- The AtomicClient updates the ITSO.ORDER table in DB2.
- WebSphere Application Server sends the placeOrder SOAP message containing the order to CICS.
- CICS uses the Web services provider PIPELINE definition PIPE1 to process the SOAP message. PIPE1 contains our SNIFFER program and the CICS-supplied SOAP 1.1 message handler DFHPISN1.
- DFHPISN1 links to the CICS-supplied header processing program DFHWSATH.
- CICS converts the SOAP message to a COMMAREA for the sample catalog manager program, DFH0XCMN.
- DFH0XCMN passes the data in the COMMAREA to the sample catalog program DFH0XVDS, which updates the recoverable VSAM file.
- DFH0XCMN passes the data in the COMMAREA to DFH0XWOD
- DFH0XWOD invokes the dispatchOrder Web service, which points to the PIPELINE PIPE2
- PIPE2 contains DFHPISN1, which:
  - Converts the COMMAREA to a SOAP message body
  - Calls DFHWSATH to create a CoordinationContext SOAP header
  - Sends the dispatchOrder SOAP message to WebSphere
- DispatchOrderAtomic updates the ITSO.DISPATCH table in DB2.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000512

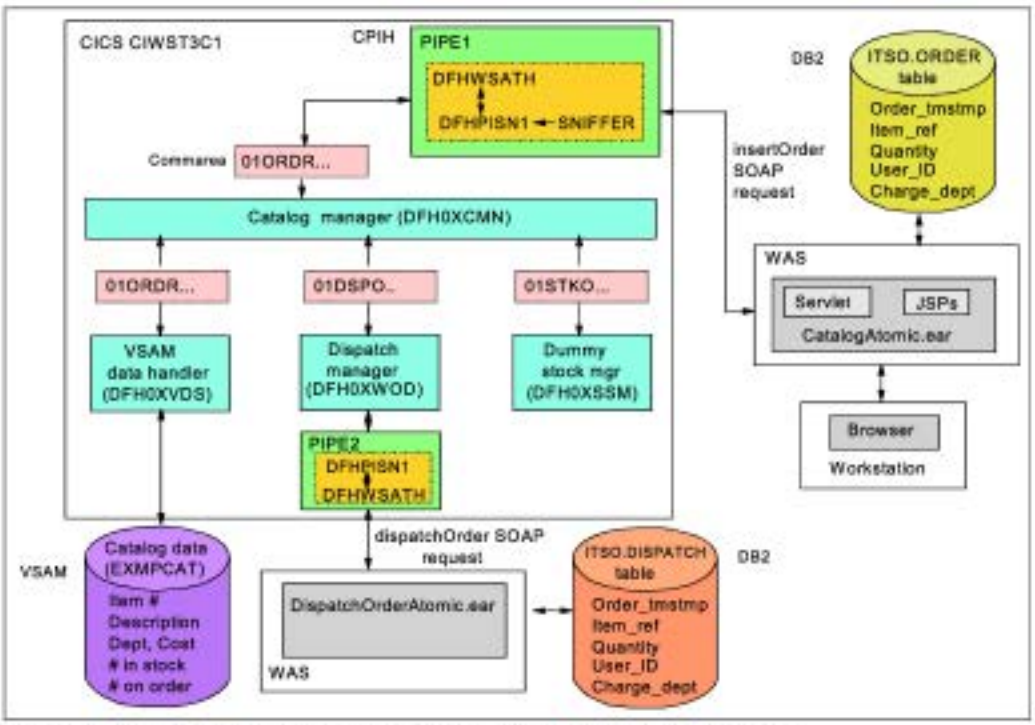


Figure 12-26 Daisy chain: CICS as a service provider and as a service requester

In the following sections we describe how we set up CICS for the daisy chain scenario, and how we created the DispatchOrderAtomic application and the ITSO.DISPATCH table. We then show the results of testing this scenario.

12.3.1 Setting up CICS for the daisy chain scenario

To set up CICS for this scenario we performed the following steps in addition to those shown in 12.2.1, "Setting up CICS for the simple scenario" on page 441:

- We edited PIPE2's configuration file /CIWS/T3C1/config/ITSO_7206_wsat_soap11request.xml

  It now contains the XML shown in Example 12-20.

Example 12-20 PIPE2:/CIWS/T3C1/config/ITSO_7206_wsat_soap11request.xml

```
<?xml version="1.0" encoding="EBCDIC-CP-US"?>
<requester_pipeline
      xmlns="http://www.ibm.com/software/htp/cics/pipeline"
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000513

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
    requester.xsd ">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler>
        <header_programs>
          <program_name>D-HWSA-H</program_name>
          <namespace>
             http://schemas.xmlsoap.org/ws/2004/10/wscoor
          </namespace>
          <localname>CoordinationContext</localname>
          <mandatory>true</mandatory>
        </header_programs>
      </cics_soap_1.1_handler>
    </service_handler_list>
  </service>
  <default_transport_handler_list>
    <handler>
      <program>SHIFTCA</program>
      <handler_parameter_list/>
    </handler>
  </default_transport_handler_list>
  <service_parameter_list>
    <registration_service_epr_address>
       http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
    </registration_service_epr_address>
  </service_parameter_list>
</requester_pipeline>
```

- We modified the configuration file for the sample Catalog application using the ECHO transaction at a 3270 terminal connected to our CICS region (see Figure 3-17 on page 98). In the 3270 screen titled Configure CICS Example Catalog Application:
  - We set the value of the Outbound WebService? field on this screen to **Yes**; this causes the Catalog Manager program (DFH0XCMN) to invoke the Dispatch Manager program (DFH0XWOD) rather than the Dummy Dispatch Manager program (DFH0XSOD) that we used in the simple scenario. While DFH0XSOD simply sets the return code in the COMMAREA to 0 and returns to its caller, DFH0XWOD issues an **EXEC CICS INVOKE WEBSERVICE('dispatchOrder') URI(outboundWebServiceURI)** command to make an outbound Web service call to an order dispatcher
  - We set the value of the Outbound WebService URI field to the location of the Web service that implements the order dispatcher function. We ran the dispatchOrder service on WebSphere Application Server for Windows.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000514

12.3.2 Creating DispatchOrderAtomic and the ITSO.DISPATCH table

To create the DispatchOrderAtomic application and the ITSO.DISPATCH table, we did the following:

- Created the DispatchDB JavaBean that inserts the dispatch order into the DB2 table.
- Changed the dispatchOrder method in the DispatchOrderSoapBindingImpl class in order to call the DispatchDB JavaBean.
  DispatchOrderSoapBindingImpl is the bean that serves the Web service and dispatchOrder is the method that is called when the Web service is requested.
- Changed the deployment descriptor of the DispatchOrderAtomic application so that it is executed as part of a Web Services Atomic Transaction.
- Created the ITSO.DISPATCH table.

We explain each step in detail in the following sections.

Create the DispatchDB JavaBean

In the same way that we previously created the OrderDB JavaBean, we created the DispatchDB JavaBean. This bean has a method that inserts the dispatch order into the ITSO.DISPATCH table. The inp1 parameter is an OrderBean that has all the data needed for the insert. We do not show the DispatchDB JavaBean code here because it is very similar to the OrderDB code shown in Example 12-4 on page 446.

Change the dispatchOrder method

When the dispatchOrder Web service is called, the dispatchOrder method in the DispatchOrderSoapBindingImpl Java class is called. It is responsible to perform the service and send a response. We changed this method to create an OrderBean and to perform the insert by calling the insertDispatch method in the DispatchDB JavaBean.

The dispatchOrder method is shown in Example 12-21.

Example 12-21 The dispatchOrder method

```
public com.Response.dispatchOrder.exampleApp.www.DispatchOrderResponse
   dispatchOrder(com.Request.dispatchOrder.exampleApp.www.DispatchOrderRequest
requestPart) throws java.rmi.RemoteException {
      System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder(): "+
            " ItemRef="+requestPart.getItemReferenceNumber()+
            " Quantity="+requestPart.getQuantityRequired()+
            " CustomerName="+requestPart.getCustomerId()+
            " Dept="+requestPart.getChargeDepartment());
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000515

```
        // Creating the order
        System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder() -
creating the order");
        OrderBean order = new OrderBean();
        order.setOrderTimestamp(new
Timestamp(Calendar.getInstance().getTime().getTime()));
        order.setItemRef(requestPart.getItemReferenceNumber());
        order.setQuantity(requestPart.getQuantityRequired());
        order.setUserId(requestPart.getCustomerId());
        order.setChargeDept(requestPart.getChargeDepartment());

        // Insert to the order in the database
        System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder() -
inserting the order in the database");
        DispatchDB dispatchDB = new DispatchDB();
        dispatchDB.insertDispatch(order);
        System.out.println("DispatchOrderSoapBindingImpl.dispatchOrder() - after
the insert!!!");

        com.Response.dispatchOrder.example5pp.www.DispatchOrderResponse
        response = new
com.Response.dispatchOrder.example5pp.www.DispatchOrderResponse();
        response.setConfirmation("Order in Dispatch");
          return response;
    }
```

Change the deployment descriptor

In order to specify that the dispatchOrder service should participate in the global transaction, we needed to modify the Web Application deployment descriptor.

To activate WS-AT support:

- We imported the application archive DispatchOrderAtomic.ear into RAD. We then expanded the **Dynamic Web Project** (DispatchOrderAtomicWeb) in the Project Explorer and opened (double-clicked) the **Deployment Descriptor** (Figure 12-27)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000516

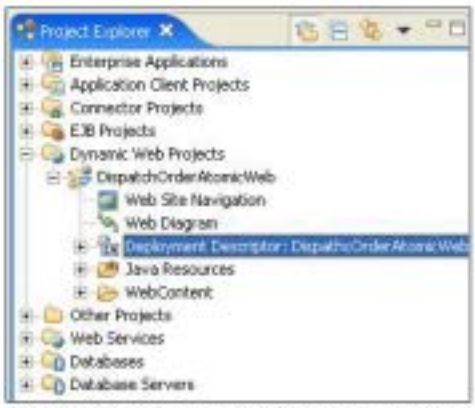


Figure 12-27 Opening the DispatchOrderAtomicWeb Deployment Descriptor

- In the Deployment Descriptor we clicked the **Servlets** tab and then we selected the **com_dispatchOrder_exampleApp_www_DispatchOrderSoapBindingImpl** servlet. Then we scrolled down to Global Transaction and selected **Execute using Web Services Atomic Transaction on incoming request** (Figure 12-28).

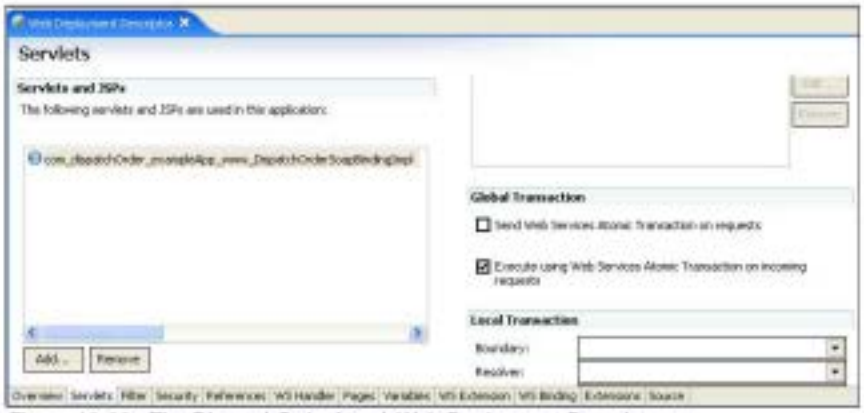


Figure 12-28 The DispatchOrderAtomicWeb Deployment Descriptor

- We saved and closed the file.

Now when an incoming request contains a coordination context, the dispatchOrder Web service will join the global transaction.

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineHistory.net
Page 517/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000517

Create the ITSO.DISPATCH table

We created the ITSO.DISPATCH table to record all the dispatch orders that come to the dispatchOrder Web service. Example 12-22 shows the script we used to create the table.

Example 12-22 Creation of ITSO.DISPATCH table

```
   DB2 CLP

CONNECT RESET;

CONNECT TO ITSODB;

-- Table definitions for Dispatch
CREATE TABLE ITSO.DISPATCH
  (DISPATCH_TMSTMP TIMESTAMP  NOT NULL ,
   ITEM_REF INTEGER  NOT NULL ,
   QUANTITY INTEGER  NOT NULL ,
   USER_ID CHARACTER (20)  NOT NULL ,
   CHARGE_DEPT CHARACTER (20)  NOT NULL  ,
   CONSTRAINT DISPATCHKEY PRIMARY KEY (DISPATCH_TMSTMP)  ) ;
```

The ITSO.DISPATCH table has a column for every field in the Catalog Place Order JSP. Also, there is a TimeStamp column that is used as a unique key for the table.

12.3.3 Testing the daisy chain scenario

We performed two tests of the daisy chain scenario:

- Normal transaction termination
- Abnormal transaction termination (see 'Daisy chain scenario: abnormal transaction termination' on page 495)

Daisy chain scenario: normal transaction termination

In this section, we explain how we ran the AtomicClient application and then show the registration and protocol service messages that are exchanged between WebSphere Application Server and CICS during the normal termination of the atomic transaction.

To run the scenario we did the following:

- Opened the welcome page of the Catalog application.

  http://cam21-pc11:9080/CatalogAtomicWeb/welcome.jsp

- Clicked **INQUIRE** to perform an inquireSingle operation.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000518

- In the **Inquire Single** window we used the Item Reference Number default value of 0010 and clicked **SUBMIT**. The Web service request was sent to CICS and we were presented with the results of the inquiry as shown in Figure 12-29.



Figure 12-29 Catalog Application - Inquire single before

- We noted that the number of items in stock was 31. This value is taken from the CICS VSAM file.
- We clicked **SUBMIT** to go to the Enter Order Details window shown in Figure 12-30.



Figure 12-30 Catalog Application - Enter order details

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net
Page 519/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000519

- In the Enter Order Details window we provided a User Name and a Department Name and clicked **SUBMIT**.
- After the CICS Web service processed the order and called the dispatchOrder WebSphere Web service, we got the response telling us that the order was successfully placed (Figure 12-31).



Figure 12-31 Catalog Application - Order successfully placed

- In the WebSphere server log we see trace entries generated by the CatalogController servlet, which show that the transaction was successfully committed (Example 12-23).

Example 12-23 WebSphere server log with successful Place Order for CatalogController

```
CatalogContoller:doPost() - Action = Place Order
CatalogController.handlePlaceOrder() - creating the order
CatalogController.handlePlaceOrder() - beginning the transaction
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - commit the transaction
CatalogController.handlePlaceOrder() - after commit
```

- We also can see the log for the WebSphere Application Server on which the DispatchOrderAtomic application is installed (Example 12-24).

Example 12-24 WebSphere server log with successful Place Order for DispatchOrder

```
DispatchOrderSoapBindingImpl.dispatchOrder(): ItemRef=10 Quantity=1
CustomerName=Luis     Dept=Itso
DispatchOrderSoapBindingImpl.dispatchOrder() - creating the order
DispatchOrderSoapBindingImpl.dispatchOrder() - inserting the order in the
database
```


Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 520/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000520

```
DispatchDB.insertDispatch() - inserted the order in the database!!!!
DispatchOrderSoapBindingImpl.dispatchOrder() - after the insert!!!
```

- Next we checked the same item number through the inquireService service and verified that the stock level decreased by one item (Figure 12-32).



Figure 12-32 Catalog Application - Inquire single after

- We opened the DB2 Control Center and used the following SQL command on the Windows machine which hosted the ITSO.ORDER table:

  ```
  SELECT * FROM ITSO.ORDER
  ```

  Figure 12-33 shows the new record in the table.

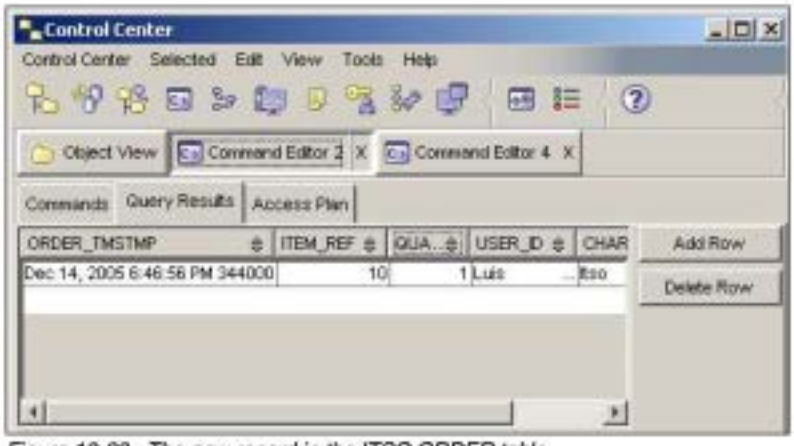


Figure 12-33 The new record in the ITSO.ORDER table

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net
Page 521/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000521

- We also used the following SQL command on the Windows machine which hosted the ITSO.DISPATCH table:

  ```
  SELECT * FROM ITSO.DISPATCH
  ```

  Figure 12-34 shows the new record in the table.

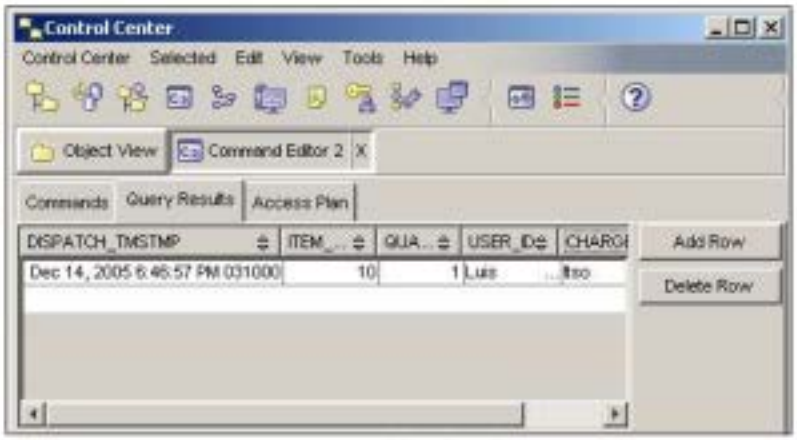


Figure 12-34 The new record in the ITSO.DISPATCH table

Figure 12-35 shows the registration and protocol service messages that are exchanged between CICS and the two WebSphere Application Servers during our test.

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineHistory.net
Page 522/8

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000522

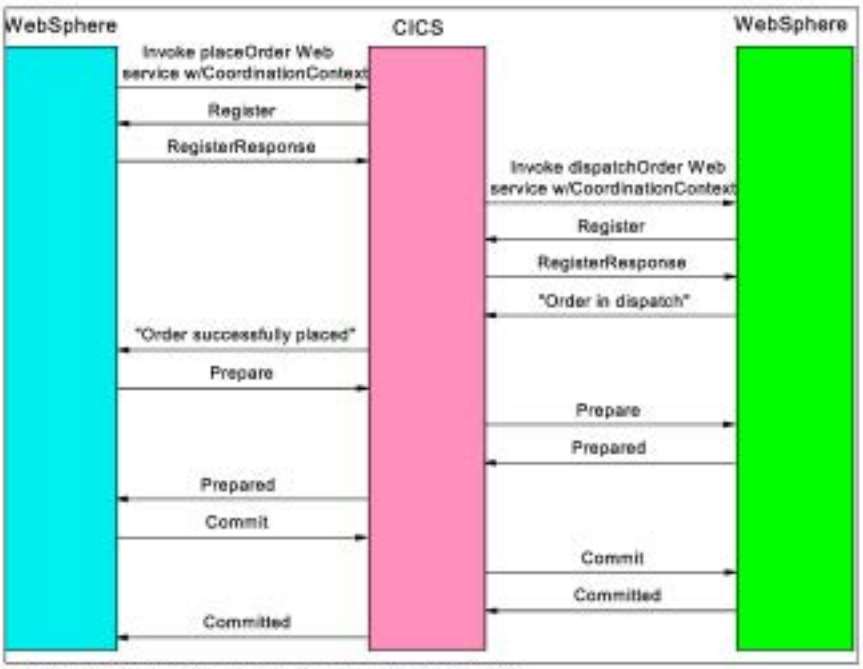


Figure 12-35 Successful daisy chain - Message flow

Example 12-25 shows the dispatchOrder request that CICS sends to WebSphere. Recall that when CICS is a *participant* in a WS-AT transaction, it uses two reference properties: `UOWID` and `PublicID`. We see in Example 12-25 that when CICS is the *coordinator*, it uses three reference properties: `UOWID`, `Token`, and `Netname`.

Example 12-25 CICS sends dispatchOrder request to WebSphere

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="..." xmlns:wscoor="..." xmlns:wsa="..."
xmlns:cicswsat="..."                   xmlns:soap="...">
   <SOAP-ENV:Header>
      <wscoor:CoordinationContext>
         <wscoor:Identifier>
            PIAT-CCON-A6POT3C1-003343578075119C
         </wscoor:Identifier>
         <wscoor:CoordinationType>
            http://schemas.xmlsoap.org/ws/2004/10/wsat
         </wscoor:CoordinationType>
         <wscoor:RegistrationService>
```


Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000523

```
            <wsa:Address>
               http://MVSG3.mop.ibm.com:15301/cicswsat/RegistrationService
            </wsa:Address>
            <wsa:ReferenceProperties>
               <cicswsat:Netname>A6P0T3C1</cicswsat:Netname>
               <cicswsat:Token>F0F0F0F0</cicswsat:Token>
               <cicswsat:UOWID>BE0F69B097751067</cicswsat:UOWID>
            </wsa:ReferenceProperties>
         </wscoor:RegistrationService>
      </wscoor:CoordinationContext>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <dispatchOrderRequest xmlns="http://www.exampleApp.dispatchOrder.Request.com">
         <itemReferenceNumber>10</itemReferenceNumber>
         <quantityRequired>1</quantityRequired>
         <customerId>Luis     </customerId>
         <chargeDepartment>Itso     </chargeDepartment>
      </dispatchOrderRequest>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Daisy chain scenario: abnormal transaction termination

Finally, we tested the daisy chain scenario, but this time with a RemoteException thrown at the end of the execution. We returned to the Place Order window to place a new order. This time we used ROLLBACK as the User Name (see Figure 12-22 on page 490). In the daisy chain scenario, this exception takes place after the insertions in the two DB2 tables and the update of the CICS VSAM file.

When the exception is thrown, the transaction is rolled back and the updates are backed out from:

- The ITSO.ORDER table
- The CICS VSAM file
- The ITSO.DISPATCH table

We can see in the WebSphere logs that the AtomicClient application successfully inserted the order record in the database prior to rolling back the transaction (Example 12-26).

Example 12-26 Catalog Application log

```
Catalogcontroller:doPost() - Action = Place Order
Catalogcontroller.handlePlaceOrder() - creating the order
Catalogcontroller.handlePlaceOrder()  beginning the transaction
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000524

```
CatalogController.handlePlaceOrder() - inserting the order in the database
OrderDB.insertOrder() - inserted the order in the database!!!!
CatalogController.handlePlaceOrder() - calling the CICS web service
CatalogController.handlePlaceOrder() - response back from the CICS web service
CatalogController.handlePlaceOrder() - simulating the RemoteException
CatalogController.handlePlaceOrder() - rollingback the transaction
```

We also can see that the DispatchOrderAtomic application successfully inserted the order record (Example 12-27).

Example 12-27 DispatchOrder Application log

```
DispatchOrderSoapBindingImpl.dispatchOrder(): ItemRef=10 Quantity=1
CustomerName=ROLLBACK Dept=Itso
DispatchOrderSoapBindingImpl.dispatchOrder() - creating the order
DispatchOrderSoapBindingImpl.dispatchOrder() - inserting the order in the
database
DispatchDB.insertDispatch() - inserted the order in the database!!!!
DispatchOrderSoapBindingImpl.dispatchOrder() - after the insert!!!
```

Following the RemoteException, the atomic transaction is rolled back. When we inquired on the stock level, we noted that there were still 30 items in stock.

12.4 Transaction scenario summary

These test scenarios demonstrate how you can synchronize WebSphere and CICS updates using WS-AT. They show how the classical 2PC distributed transaction can be implemented using Web services, and that the distributed global transaction can be committed or rolled back based on a set of Web service flows that are managed entirely by the WebSphere and CICS middleware.

Important: Before implementing a solution based on WS-AT, you should be aware of the general issues that can arise from any implementation of a distributed transaction (for example, locked records preventing access to important data) and you should also compare the solution with alternatives such as the J2EE Connector Architecture.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000525

Part 5

Appendixes

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000526

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000527

A

Sample handler programs

In this appendix we show the sample message handler and header processing programs used in our test scenarios:

- Sample message handler program - CIWSMSGH
- Sample header processing program - CIWSSECH
- Sample handler program - SNIFFER
- Sample XML parser program - MYPARSER
- Sample header processing program - CIWSSECR
- Sample header processing program - CIWSSECS
- Sample header processing program - WSATHND

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000528

A.1 Sample message handler program - CIWSMSGH

The CIWSMSGH program in Example A-1 was used to change the transaction ID for the Web service requests received in the pipeline.

The program obtains the value in the DFHFUNCTION container and if the program was invoked for a RECEIVE-REQUEST function, then it continues processing. It then obtains the data in the DFHWS-WEBSERVICE container and makes a decision based on the contents of the requested Web service name as to which transaction ID should be used. This new value for the transaction ID is put into the DFHWS-TRANID container. The program puts the contents of the DFHWS-TRANID and the DFHWS-WEBSERVICE containers to the CESE CICS transient data queue.

Example A-1 Sample message handler program - CIWSMSGH

```
CC01CC IDENTIFICATION DIVISION.
CC02CC PROGRAM-ID.  CIWSMSGH
CC03CC*****************************************************************
CC04CC*                                                               *
CC05CC* This program:                                                 *
CC06CC* 1. CHANGES THE DEFAULT TRANID : CPIH/CPIL                     *
CC07CC*    BASED ON THE INCOMING SOAP REQUESTS                        *
CC08CC*****************************************************************
CC09CC
CC10CC AUTHOR.      CHEREGATTI.
CC11CC DATE-COMPILED.
CC12CC ENVIRONMENT DIVISION.
CC13CC CONFIGURATION SECTION.
CC14CC SPECIAL-NAMES.
CC15CC DATA DIVISION.
CC16CC WORKING STORAGE SECTION.
CC17CC 01 WS-START.
CC18CC* Nesting offset for DISPLAYS
CC19CC   03 NN                    PIC X(11) VALUE 'CIWSMSGH: '.
CC20CC
CC21CC   03 RESP                     PIC S9(8) COMP-5 SYNC.
CC22CC   03 RESP2                    PIC S9(8) COMP-5 SYNC.
CC23CC   03 HOH-PTR                  USAGE IS POINTER.
CC24CC   03 HOH-LEN                  PIC S9(8) COMP-4.
CC25CC   03 CONTAINER-LEN            PIC S9(8) BINARY.
CC26CC   03 GETMAIN-PTR              USAGE IS POINTER.
CC27CC   03 GETMAIN-LEN              PIC S9(8) COMP-4.
CC28CC   03 WS-FAULT-STRING          PIC X(21) value spaces.
CC29CC   03 WS-DFHFUNCTION           PIC X(16) value spaces.
CC30CC   03 WS-WEBSERVICES           PIC X(32) value spaces.
CC31CC   03 WS-WEBSERVICES-LEN       PIC S9(8) BINARY
CC32CC*
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000529

```
CC33OC* Not found SOAP Fault Detail section
CC34OC*
CC35OC 01  WS-Fault-NotFnd.
CC36OC   03 WS-Fault-Namespace   pic x(53).
CC37OC   03 WS-Fault-OC-Lit      pic X(7).
CC38OC   03 WS-Fault-OC          pic X(2).
CC39OC   03 WS-Fault-RC-Lit      pic X(10).
CC40OC   03 WS-Fault-Item-Lit    pic X(11).
CC41OC   03 WS-Fault-Item        pic X(4).
CC42OC   03 WS-Fault-Item-Lit    pic X(12).
CC43OC   03 WS-Fault-CNamespace  pic x(19).
CC44OC
CC45OC 01 EXP4RSER-COM_LEN            PIC S9(4) COMP-4
CC46OC 01 C4-PARSER-MSG.
CC47OC    03 C4-PARSER-REQUEST.
CC48OC      05 FILLER                PIC X(2).
CC49OC      05 C4-TRANID             PIC X(4).
CC50OC    03 C4-PARSER-RESP-REQ      pic x(4).
CC51OC    03 C4 PARSER RET CODE      pic x(2).
CC52OC    03 FILLER                  PIC X(200).
CC53OC
CC54OC*****************************************************************
CC55OC* Externally referenced data areas
CC56OC*****************************************************************
CC57OC LINKAGE SECTION.
CC58OC 01 BIG-AREA.
CC59OC    02 FILLER    PIC X OCCURS 64000 DEPENDING ON BIG-LEN.
CC60OC
CC61OC 01 GETMAIN AREA.
CC62OC    02 FILLER    PIC X OCCURS 64000 DEPENDING ON GETMAIN LEN.
CC63OC 01 CONTAINER-DATA.
CC64OC    05 FILLER            PIC X OCCURS 32768
CC65OC                           DEPENDING ON CONTAINER-LEN.
CC66OC*----------------------------------------------------------------*
CC67OC    EJECT
CC68OC*----------------------------------------------------------------*
CC69OC PROCEDURE DIVISION.
CC70OC*                                                                *
CC71OC     EXEC CICS GET CONTAINER('DFHFUNCTION')
CC72OC                   INTO(WS-DFHFUNCTION)
CC73OC                   FLENGTH(length of WS-DFHFUNCTION)
CC74OC                   NOHANDLE
CC75OC     END-EXEC.
CC76OC* If not RECEIVE-REQUEST then exit
CC77OC     IF WS-DFHFUNCTION not = 'RECEIVE-REQUEST
CC78OC         PERFORM VALIDATE-REQUEST THRU END-VALIDATE-REQUEST
CC79OC         PERFORM CHANGE-TRANID    THRU END-CHANGE-TRANID
CC80OC         EXEC CICS
CC81OC            DELETE CONTAINER('DFHRESPONSE')
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000530

```
CC82CC         END-EXEC
CC83CC     END-IF
CC84CC     EXEC CICS RETURN END-EXEC.
CC85CC     GOBACK.
CC86CC*---------------------------------------------------------------
CC87CC     EJECT
CC88CC*---------------------------------------------------------------
CC89CC*              REQUEST VALIDATION
CC90CC*
CC91CC VALIDATE-REQUEST.
CC92CC     PERFORM GET-SOAP-WEBSERVICE THRU END-GET-SOAP-WEBSERVICE.
CC93CC     IF WS-WEBSERVICES = 'ERROR'
CC94CC         PERFORM FAULT-INVREQ
CC95CC     END-IF.
CC96CC END-VALIDATE-REQUEST. EXIT.
CC97CC*---------------------------------------------------------------
CC98CC     EJECT
CC99CC*---------------------------------------------------------------
C10CCC* Retrieve the C-HWS WEBSERVICE that contains type of requests
C101CC*---------------------------------------------------------------
C102CC GET-SOAP-WEBSERVICE.
C103CC     EXEC CICS
C104CC     GET CONTAINER('DFHWS-WEBSERVICE')
C105CC     SET(ADDRESS OF CONTAINER-DATA)
C106CC     FLENGTH(CONTAINER-LEN)
C107CC     END-EXEC.
C108CC* Copy the input container to our storage
C109CC     MOVE CONTAINER-DATA(1:50) TO WS-WEBSERVICES.
C110CC     MOVE CONTAINER-LEN TO WS-WEBSERVICES-LEN.
C111CC END-GET-SOAP-WEBSERVICE. EXIT.
C112CC*---------------------------------------------------------------
C113CC     EJECT
C114CC*---------------- CHANGE DEFAULT TRANID OPEN/CICS ---------------
C115CC CHANGE-TRANID.
C116CC       EXEC CICS GET CONTAINER('DFHWS-TRANID')
C117CC            SET(ADDRESS OF CONTAINER-DATA)
C118CC            FLENGTH(CONTAINER-LEN)
C119CC       END EXEC.
C120CC     IF WS-WEBSERVICES = 'inquireSingle'
C121CC        MOVE 'INQS' TO CA-TRANID
C122CC        PERFORM CHANGE-CONTAINER THRU END-CHANGE-CONTAINER
C123CC     END-IF
C124CC     IF WS-WEBSERVICES = 'inquireCatalog'
C125CC        MOVE 'INQC' TO CA-TRANID
C126CC        PERFORM CHANGE-CONTAINER THRU END-CHANGE-CONTAINER
C127CC     END-IF
C128CC     IF WS-WEBSERVICES   'placeOrder'
C129CC        MOVE 'ORDR' TO CA-TRANID
C130CC     END IF.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000531

```
C1291C     IF WS-WEBSERVICES = 'dispatch0-devEndpoint'
C1292C        MOVE 'DISP' TO CA-TRANID
C1293C        PERFORM CHANGE-CONTAINER THRU END-CHANGE-CONTAINER
C1310C     END-IF.
C1320C END-CHANGE-TRANID. EXIT.
C1330C*---------------------------------------------------------------
C1340C     EJECT
C1350C*
C1360C CHANGE CONTAINER.
C1370C     MOVE CA-TRANID TO CONTAINER-DATA(1:4)
C1380C     EXEC CICS PUT CONTAINER('DFHWS-TRANID')
C1390C          FROM(CONTAINER-DATA)
C1400C          FLENGTH(CONTAINER-LEN)
C1410C     END-EXEC.
C1420C     DISPLAY MM '+--------------------------------+'
C1430C     DISPLAY MM 'Container Name: : DFHWS-WEBSERVICE '.
C1440C*    DISPLAY MM 'Content length: '    WS-WEBSERVICES-LEN.
C1450C     DISPLAY MM 'Container contents: ' WS-WEBSERVICES.
C1460C     DISPLAY MM '                    '.
C1470C     DISPLAY MM 'Container Name: : DFHWS-TRANID  .
C1480C*    DISPLAY MM 'Content length: '    CONTAINER-LEN
C1490C     DISPLAY MM 'Container content: ' CONTAINER-DATA.
C1500C END-CHANGE-CONTAINER  EXIT.
C1510C     EJECT
C1520C*****************************************************************
C1530C* We detected that the ca request id field specifies an invalid
C1540C* request. This is a CLIENT error.
C1550C*****************************************************************
C1560C FAULT_INVREQ SECTION.
C1570C*                                                               *
C1580C* Generate a SOAP Fault
C1590C*---------------------------------------------------------------*
C1600C     MOVE 'Request code invalid' TO WS-FAULT-STRING
C1610C     EXEC CICS SOAPFAULT CREATE
C1620C               FAULTCODE(d'invalue|CLIENT|)
C1630C               FAULTSTRING(WS-FAULT-STRING)
C1640C               FAULTSTRLEN(length of WS-FAULT-STRING)
C1650C     END EXEC.
C1660C FAULT_INVREQ_END. EXIT.
C1670C*****************************************************************
C1680C* The supplied ca_item_req_ref reference is not in our database
C1690C* We decide to send a SOAP fault.
C1700C* This is a SERVER error
C1710C* We supply detailed information in the DETAIL fault element
C1720C*****************************************************************
C1730C FAULT-NOTFND SECTION.
C1740C*---------------------------------------------------------------*
C1750C* Build the Detail section.
C1760C*  we do it this way for pedagogical reasons
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000532

```
C1770C*-------------------------------------------------------------*
C1780C     MOVE '<o(   NOT FOUND   )o>' to WS-FAULT-STRING
C1790C     MOVE
C1800C     '<hdlh:FaultData ' xmlns:hdlh="http://HDR40LR2.fault">'
C1810C                              to WS-Fault-Namespace.
C1820C     MOVE '<hdlh:ReturnCode>' to WS-Fault-RC-Lit.
C1830C     MOVE DFHPARSER-RETURNCODE to WS-Fault-RC.
C1840C     MOVE '</hdlh:ReturnCode>' to WS-Fault-RC-Lit.
C1850C     MOVE '<hdlh:Item>'        to WS-Fault-Item-Lit.
C1860C     MOVE DFHPARSER-REF-REQ    to WS-Fault-Item.
C1870C     MOVE '</hdlh:Item>'       to WS-Fault-Item-Lit.
C1880C     MOVE '</hdlh:FaultDetail>'
C1890C            to WS-Fault-ENamespace.
C1900C     EXEC CICS SOAPFAULT CREATE
C1910C               DETAIL(WS-Fault-NotFnd)
C1920C               DETAILLENGTH(length of WS-Fault-NotFnd)
C1930C               FAULTCODE(DFHVALUE(SERVER))
C1940C               FAULTSTRING(WS-FAULT-STRING)
C1950C               FAULTSTRLEN(length of WS-FAULT-STRING)
C1960C     END-EXEC.
C1970C FAULT-INVREQ-EXC. EXIT.
```

Sample output from CIWSMSGH

Example A-2 shows the output sent to the CESE transient data queue by the CIWSMSGH message handler program, whenever it changes the transaction ID in the DFHWS-TRANID container.

Example A-2 CIWSMSGH - sample output

```
CIWSMSGH:  ----------------------------------
CIWSMSGH:  Container Name: - DFHWS-WEBSERVICE
CIWSMSGH:  Container Content- InquireCatalog
CIWSMSGH:  ----------------------------------
CIWSMSGH:  Container Name: - DFHWS-TRANID
CIWSMSGH:  Container Content: CIQC
CIWSMSGH:  >                                <
CIWSMSGH:  Container Name: - DFHWS-WEBSERVICE
CIWSMSGH:  Container Content- InquireSingle
CIWSMSGH:
CIWSMSGH:  Container Name: - DFHWS-TRANID
CIWSMSGH:  Container Content: CIQS
CIWSMSGH:  >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
CIWSMSGH:  Container Name: - DFHWS-WEBSERVICE
CIWSMSGH:  Container Content- placeOrder
CIWSMSGH:  ----------------------------------
CIWSMSGH:  Container Name: - DFHWS-TRANID
CIWSMSGH:  Container Content- ORDR
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000533

A.2 Sample header processing program - CIWSSECH

The CIWSSECH program (Example A-3) extracts security credentials from the WS-Security header, validates them and changes the user ID under which the business logic of the invoked Web service executes

- It first determines that it is invoked during the RECEIVE-REQUEST phase of processing.
- It then ensures that this is only executed for a "place an order" request by examining the data in the DFHWS-URI container. In the event that this is not the case, it returns control to the calling program.
- When it is processing an order request, the WS-Security header is obtained from the DFHHEADER container.
- The header is then passed in a COMMAREA to a COBOL XML parsing program that extracts the user ID and password from the XML.
- CIWSSECH then uses the EXEC CICS VERIFY PASSWORD command to determine if the user ID and password are correct. If this is true, then the user ID is written to the DFHWS-USERID container, which will cause a context switch to occur when invoking the business logic. If the verification returns an error, then a SOAP fault is created and returned to the requester.

Example. A-3 Sample header processing program: CIWSSECH

```
CC01CC  PROCESS CICS
CC02CC IDENTIFICATION DIVISION.
CC03CC PROGRAM-ID.  CIWSSECH
CC04CC*****************************************************************
CC05CC*                                                               *
CC06CC* This program:                                                 *
CC07CC* 1. Checks for RECEIVE REQUEST function, else returns          *
CC08CC* 2. Obtains data from DFHHEADER container                       *
CC09CC* 3. Uses the PYPARSER program to extract the CA-USER-ID        *
CC10CC*                                    and the CA-PASSWORD        *
CC11CC*    fields from the XML message contained within the Body      *
CC12CC* 4. Verifies the user ID/password combination                   *
CC13CC*    If it is successful, then it places the CA-USER-ID into    *
CC14CC*        the DFHWS-USERID container.                            *
CC15CC*    If it fails, it returns a SOAP fault message in DFHHESHDSER*
CC16CC*****************************************************************
CC17CC
CC18CC AUTHOR.      CICSML.
CC19CC DATE-COMPILED.
CC20CC ENVIRONMENT DIVISION.
CC21CC CONFIGURATION SECTION
CC22CC SPECIAL-NAMES.
CC23CC DATA DIVISION.
CC24CC WORKING-STORAGE SECTION.
CC25CC 01 WS-STUFF.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000534

```
002600   03 FILLER                  PIC X(44)
002700       VALUE '*** CLASSIC WORKING STORAGE STARTS HERE ***'.
002800   03 WS-RESP                  PIC S9(8) COMP-5 SYNC.
002900   03 WS-RESP2                 PIC S9(8) COMP-5 SYNC.
003000   03 WS-HEAD-PTR              USAGE IS POINTER.
003100   03 WS-HEAD-LEN              PIC S9(8) COMP-4.
003200   03 WS-URI-PTR               USAGE IS POINTER.
003300   03 WS URI LEN               PIC S9(8) COMP 4.
003400   03 WS FUNC LEN              PIC S9(8) COMP 4 VALUE 16.
003500   03 WS GETPATH PTR           USAGE IS POINTER.
003600   03 WS-GETPATH-LEN           PIC S9(8) COMP-4.
003700   03 WS-FAULT-STRING          PIC X(100) value spaces.
003800   03 WS-FAULT-CODE            PIC S9(8) COMP-4
003900   03 WS-SOAP-LEVEL            PIC S9(8) COMP-4.
004000     88 WS-SOAP-11  VALUE 1.
004100     88 WS-SOAP-12  VALUE 2.
004200     88 WS-NOT-SOAP VALUE 10.
004300   03 WS-FUNC-AREA             PIC X(16).
004400   03 WS NOT AUTH              PIC X(40)
004500               VALUE  Not authorized to place order.'.
004600   03 WS-AUTH-FAILED           PIC X(40)
004700               VALUE  Authorization failed for order request.'.
004800
004900 01 MYPARSER-COM_LEN            PIC S9(4) COMP-4
005000 01 DFH-PARSER-OBJ.
005100    03 DFH-USER-ID             PIC X(8).
005200    03 DFH-PASSWORD            PIC X(8).
005300
005400 LINKAGE SECTION.
005500 01 WS HEAD AREA.
005600    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-HEAD-LEN.
005700
005800 01 WS-URI-AREA.
005900    02 FILLER    PIC X OCCURS 256 DEPENDING ON WS-URI-LEN.
006000
006100 01 WS-GETPATH-AREA.
006200    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-GETPATH-LEN.
006300
006400*****************************************************************
006500* Main line code begins                                         *
006600*****************************************************************
006700 PROCEDURE DIVISION.
006800 MAIN-PROCESSING SECTION.
006900
007000*****************************************
007100*  Receive the SOAP Body namespace
007200*****************************************
007300     PERFORM GET-DFHFUNCTION.
007400
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000535

```
007500*********************************
007600*  Receive the SOAP Body namespace
007700*********************************
007800     PERFORM GET-DFHWS-URI.
007900     IF WS-URI-AREA(1:WS-URI-LEN)
008000              NOT =  '/example4pp/placeOrder'
008100        EXEC CICS RETURN END-EXEC
008200     END-IF.
008300
008400*********************************
008500*  Receive the SOAP Body namespace
008600*********************************
008700     PERFORM GET-SOAP-HEADER.
008800
008900*****************************************************************
009000*  The SOAP body XML data can now be parsed by MYPARSER
009100*    LINK on the XML parser, it will return the CA-USER-ID
009700*                                          and CA-PASSWORD
009200*                                          or   'EXCEPTION'
009400*                                          or   'NOT FOUND'
009500*                                          or   'BAD COMMA'
009600*****************************************************************
009700     EXEC CICS LINK PROGRAM('MYPARSER')
009800                    COMMAREA(WS-HEAD-AREA)
009900                    LENGTH(MYPARSER-COMPLEN)
010000     END-EXEC.
010100     IF WS-HEAD-AREA(1:9)   'EXCEPTION' or
010700                            'NOT FOUND' or
010200                            'BAD COMMA'
010400*****************************************************
010500*  Error found during the XML PARSE program execution
010600*****************************************************
010700         MOVE WS-ALTH-FAILED TO WS-FAULT-STRING
010800         PERFORM FAULT-MESSAGE
010900     ELSE
011000         MOVE WS-HEAD-AREA to CA-PARSER-RESP
011100*****************************************
011200*  OK, data found, so do the VERIFY
011300*****************************************
011400         EXEC CICS VERIFY
011500            PASSWORD(CA-PASSWORD)
011600            USERID(CA-USER-ID)
011700            RESP(WS-RESP)
011800         END-EXEC
011900*****************
012000*  Successful V
012100*****************
012700         IF WS-RESP   DFHRESP(NORMAL)
012200            MOVE CA-  USER-ID
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000536

```
C12400         ELSE
C12500            MOVE WS-NOT-AUTH TO WS-FAULT-STRING
C12600            PERFORM FAULT-MESSAGE
C12700         END-IF
C12800     END-IF.
C12900*************************
C13000* and is the end, bye bye
C13100*************************
C13200     EXEC CICS RETURN END EXEC.
C13300
C13400 MAIN-PROCESSING-END. EXIT.
C13500
C13600*****************************************************
C13700*     ***********************************     *
C13800*              SUBROUTINES FOLLOW                 *
C13900*     ***********************************     *
C14000*****************************************************
C14100
C14200*****************************************************************
C14300* Retrieve the function type from DFHFUNCTION container
C14400*****************************************************************
C14500 GET-DFHFUNCTION.
C14600     EXEC CICS
C14700        GET CONTAINER( DFHFUNCTION')
C14800        INTO(WS-FUNC-AREA)
C14900        FLENGTH(WS-FUNC-LEN)
C15000        RESP(WS-RESP)
C15100     END-EXEC.
C15200
C15300*************************************************
C15400* Check for correct length. Create SOAP fault if
C15500* not correct. If for incoming request then we
C15600* do the work, else we exit immediately.
C15700*************************************************
C15800     IF WS-FUNC- Eh NOT - 15
C15900        MOVE WS-ALTH-FAILED TO WS-FAULT-STRING
C16000        PERFORM FAULT-MESSAGE
C16100     ELSE
C16200        IF WS FUNC AREA NOT = 'RECEIVE REQUEST'
C16300           EXEC CICS RETURN END EXEC
C16400        END-IF
C16500     END-IF.
C16600
C16700*****************************************************************
C16800* Retrieve the URI from the DFHWS-URI container
C16900*****************************************************************
C17000 GET-DFHWS-URI.
C17100     EXEC CICS
C17200        GET CONTAINER( DFHWS URI')
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000537

```
C17300        SET(WS-URI-PTR)
C17400        FLENGTH(WS-URI-LEN)
C17500        RESP(WS-RESP)
C17600     END-EXEC.
C17700
C17800*****************************************
C17900* Copy the input container to our storage
C18000*****************************************
C18100     IF WS URI LEN > 0
C18200        SET ADDRESS OF WS-REQ-AREA TO WS-URI-PTR
C18300        MOVE WS-URI-LEN TO WS-GETMAIN-LEN
C18400
C18500        EXEC CICS GETMAIN
C18600            SET(WS-GETMAIN-PTR)
C18700            FLENGTH(WS-GETMAIN-LEN)
C18800        END-EXEC
C18900
C19000        SET ADDRESS OF WS-GETMAIN-AREA TO WS-GETMAIN-PTR
C19100        MOVE WS URI AREA TO WS GETMAIN AREA
C19200        SET WS-URI-PTR TO WS-GETMAIN-PTR
C19300        SET ADDRESS OF WS-REQ-AREA TO WS-URI-PTR
C19400     ELSE
C19500        MOVE WS-PATH-FAILED TO WS-FAULT-STRING
C19600        PERFORM FAULT-MESSAGE
C19700        EXEC CICS RETURN END-EXEC
C19800     END-IF.
C19900
C20000*****************************************
C20100* Retrieve the <wsse:Security> header
C20200*****************************************
C20300 GET-SOAP-HEADER.
C20400     EXEC CICS
C20500        GET CONTAINER( OFFHEADER')
C20600        SET(WS-HEAD-PTR)
C20700        FLENGTH(WS-HEAD-LEN)
C20800        RESP(WS-RESP)
C20900     END-EXEC.
C21000
C21100*****************************************
C21200* Copy the input container to our storage
C21300*****************************************
C21400     IF WS-HEAD-LEN > 0
C21500        SET ADDRESS OF WS-HEAD-AREA TO WS-HEAD-PTR
C21600        MOVE WS-HEAD-LEN TO WS-GETMAIN-LEN MYPARSER-CONLEN
C21700
C21800        EXEC CICS GETMAIN
C21900            SET(WS-GETMAIN-PTR)
C22000            FLENGTH(WS-GETMAIN-LEN)
C22100        END EXEC
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000538

```
C2220C
C2230C        SET ADDRESS OF WS-GETMAIN-AREA TO WS-GETMAIN-PTR
C2240C        MOVE WS-HEAD-AREA TO WS-GETMAIN-AREA
C2250C        SET WS-HEAD-PTR TO WS-GETMAIN-PTR
C2260C        SET ADDRESS OF WS-HEAD-AREA TO WS-HEAD-PTR
C2270C     ELSE
C2280C        MOVE WS-ALTE-FAILED TO WS-FAULT-STRING
C2290C        PERFORM FAULT-MESSAGE
C2300C        EXEC CICS RETURN END EXEC
C2310C     END-IF.
C2320C
C2330C*****************************************************************
C2340C* The user ID has been verified, we will now set the contents of
C2350C* DFHWS-USERID container with this value.This will cause the
C2360C* business logic (back-end program/s) to be executed with this
C2370C* user ID, using it's access rights.
C2380C*****************************************************************
C2390C SET-USER-ID.
C2400C     EXEC CICS
C2410C         PUT CONTAINER('DFHWS-USERID')
C2420C         FROM(CA-USER-ID)
C2430C         FLENGTH(length of CA-USER-ID)
C2440C         DATATYPE(DFHVALUE(CHAR))
C2450C         RESP(WS-RESP)
C2460C     END-EXEC.
C2470C
C2480C*****************************************************************
C2490C* We detected that the ca_request_id field specifies an invalid
C2500C* request. This is a CLIENT error.
C2510C*****************************************************************
C2520C FAULT-MESSAGE SECTION.
C2530C************************
C2540C* Generate a SOAP Fault
C2550C************************
C2560C     EXEC CICS
C2570C         GET CONTAINER('DFHWS-SOAPLEVEL')
C2580C         INTO(WS-SOAP-LEVEL)
C2590C         FLENGTH(WS-READ-LEN)
C2600C         RESP(WS-RESP)
C2610C     END-EXEC.
C2620C
C2630C*************************************
C2640C* MOVE CORRECT VERSION OF FAULTCODE
C2650C*************************************
C2660C     IF WS-SOAP-11 MOVE DFHVALUE(CLIENT) TO WS-FAULT-CODE
C2670C     ELSE
C2680C        MOVE DFHVALUE(SENDER) TO WS-FAULT-CODE
C2690C     END-IF
C2700C
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000539

```
027100     EXEC CICS SCAFFAULT CREATE
027200               FAULTSTRING(WS-FAULT-STRING)
027300               FAULTSTRLEN(LENGTH OF WS-FAULT-STRING)
027400               FAULTCODE(WS-FAULT-CODE)
027500     END-EXEC.
027600 FAULT-MESSAGE-END. EXIT.
```

A.3 Sample handler program - SNIFFER

The SNIFFER handler program can be used as a message handler program or a header processing program.

It is a very simple program that browses through the containers available in the pipeline. It does this by issuing a STARTBROWSE CONTAINER command followed by GETNEXT CONTAINER until all have been browsed, then it issues an ENDBROWSE CONTAINER command. For each container that it issues a GETNEXT CONTAINER for, it displays the name and contents to the CICS transient data queue CESE

Example A-4 Sample handler program - SNIFFER

```
000100CBL TRUNC(OPT)
000200 IDENTIFICATION DIVISION.
000300 PROGRAM-ID. SNIFFER.
000400*****************************************************************
000500* HEADER START                                                  *
000600*                                                               *
000700*                                                               *
000800* PROGRAM NAME: See above                                       *
000900*                                                               *
001000* TITLE: Pipeline Channel container sniffer                     *
001100*                                                               *
001200* DATE: 03/06/05                                                *
001300*                                                               *
001400* AUTHOR: Ian Noble                                             *
001500*                                                               *
001600* CHANGE HISTORY:                                               *
001700*                                                               *
001800*   DATE MODIFIED  CHANGED BY        REASON FOR CHANGE          *
001900*                                                               *
002000* PROGRAM DESCRIPTION.                                          *
002100*                                                               *
002200*   This program writes a report to the CICS transient data     *
002300*   queue which is usually an extra-partition queue having a    *
002400*   DDname of CEEMSG. The report contains the name, length, and *
002500*   contents of every container which is visible to user code   *
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000540

```
CC26CC*  within a pipeline as shown in the following example:          *
CC27CC*                                                               *
CC28CC*  SNIFFER :  *** Start ***                                     *
CC29CC*  SNIFFER :  ********************************                 *
CC30CC*  SNIFFER :  Container Name   : DFHFUNCTION                    *
CC31CC*  SNIFFER :  Content length   : 00000016                       *
CC32CC*  SNIFFER :  Container content:                                *
CC33CC*  RECEIVE HEQLEST                                             *
CC34CC*  SNIFFER :  Containers on channel: List starts.               *
CC35CC*  SNIFFER :  >-------------------------------<                 *
CC36CC*  SNIFFER :  Container Name   : DFH-HANDLEPLIST                *
CC37CC*  SNIFFER :  Content length   : 00000000                       *
CC38CC*  SNIFFER :  Container EMPTY                                   *
CC39CC*  SNIFFER :  >-------------------------------<                 *
CC40CC*  SNIFFER :  Container Name   : DFHNFSFCh$F                    *
CC41CC*  SNIFFER :  Content length   : 00000000                       *
CC42CC*  SNIFFER :  Container EMPTY                                   *
CC43CC*  SNIFFER :  >                               <                 *
CC44CC*  SNIFFER :  Container Name   : UFHFUNCTION                    *
CC45CC*  SNIFFER :  Content length   : 00000016                       *
CC46CC*  SNIFFER :  Container content:                                *
CC47CC*  RECEIVE-REQUEST                                             *
CC48CC*  SNIFFER :  ********************************                 *
CC49CC*  SNIFFER :  Container Name   : DFHWS-SOAPACTION               *
CC50CC*  SNIFFER :  Content length   : 00000007                       *
CC51CC*  SNIFFER :  Container content:                                *
CC52CC*  ""                                                          *
CC53CC*  SNIFFER :  >                                   <             *
CC54CC*  SNIFFER :  Container Name   : UFHWS URL                      *
CC55CC*  SNIFFER :  Content length   : 00000022                       *
CC56CC*  SNIFFER :  Container content:                                *
CC57CC*  /exampleApp/placeOrder                                      *
CC58CC*  SNIFFER :  ********************************                 *
CC59CC*  SNIFFER :  Container Name   : DFHREQLIST                     *
CC60CC*  SNIFFER :  Content length   : 00007C94                       *
CC61CC*  SNIFFER :  Container content:                                *
CC62CC*  <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoa...    *
CC63CC*  " xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:x...    *
CC64CC*  lsoap.org/ws/2004/10/wsaddr" xmlns:wsa="http://schemas...    *
CC65CC*  etc. etc. etc. etc. etc. etc. etc. etc. etc. etc. etc.      *
CC66CC*  SNIFFER :  ********************************                 *
CC67CC*  SNIFFER :  Container Name   : DFH-SERVICEPLIST               *
CC68CC*  SNIFFER :  Content length   : 00000134                       *
CC69CC*  SNIFFER :  Container content:                                *
CC70CC*  <registration service endpoint>       http://MWS33.top etc.  *
CC71CC*  SNIFFER :  >-------------------------------<                 *
CC72CC*  SNIFFER :  Container Name   : UFHWS-PIPELINE                 *
CC73CC*  SNIFFER :  Content length   : 00000008                       *
CC74CC*  SNIFFER :  Container content:                                *
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000541

```
CC75CC*   PIPEL                                                          *
CC76CC*   SNIFFER :  ********************************                  *
CC77CC*   SNIFFER :  Container Name   : DFHWS-USERID                    *
CC78CC*   SNIFFER :  Content length   : 00000008                        *
CC79CC*   SNIFFER :  Container content:                                 *
CC80CC*   CICS3D                                                       *
CC81CC*   SNIFFER :  >                                 <               *
CC82CC*   SNIFFER :  Container Name   : DFHWS-TRANID                   *
CC83CC*   SNIFFER :  Content length   : 00000004                        *
CC84CC*   SNIFFER :  Container content:                                 *
CC85CC*   IPIC                                                         *
CC86CC*   SNIFFER :  ********************************                  *
CC87CC*   SNIFFER :  Container Name   : DFHWS-ACCESSACCE               *
CC88CC*   SNIFFER :  Content length   : 00000C32                        *
CC89CC*   SNIFFER :  Container content:                                 *
CC90CC*   placeOrder                                                   *
CC91CC*   SNIFFER :  >                                 <               *
CC92CC*   SNIFFER :  Container Name   : DFHWS-APPHANDLER                *
CC93CC*   SNIFFER :  Content length   : 00000008                        *
CC94CC*   SNIFFER :  Container content:                                 *
CC95CC*   DFHPITP                                                       *
CC96CC*   SNIFFER :  Containers on channel: List ends                  *
CC97CC*   SNIFFER :  DFHRESPONSE      Container deleted                 *
CC98CC*   SNIFFER :  **** End ****                                     *
CC99CC*                                                                 *
CCCCCC*-------------------------------------------------------------------*
C10010C
C10020C ENVIRONMENT DIVISION.
C10030C DATA DIVISION.
C10040C WORKING STORAGE SECTION.
C10050C
C10060C 77  A5-EYE-CATCHER         PIC X(44)     VALUE
C10070C                  '*** SNIFFER WORKING STORAGE STARTS HERE ***'.
C10080C
C10090C* Working offset for DISPLAYS
C10100C 77  WW                    PIC X(11) VALUE 'SNIFFER : '.
C10110C
C10120C* Miscellaneous variables
C10130C
C10140C 77 CONTAINER-FOUND         PIC X(01).
C10150C
C10160C 77 CHANNEL-NAME               PIC X(16).
C10170C 77 CONTAINER-NAME             PIC X(16).
C10180C 77 CONTAINER-LEN              PIC S9(8) COMP-4 SYNCHRONIZED
C10190C                                          VALUE 0.
C10200C
C10210C 77 DFHEIBLK-ITEM-PTR          USAGE IS POINTER.
C10220C 77 DFHEIBLK-ITEM-LEN          PIC S9(8) COMP-4 SYNCHRONIZED
C10230C                                          VALUE 0.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000542

```
012400
012500 77 GET-MAIN-PTR                USAGE IS POINTER
012600 77 GET-MAIN-LEN                PIC S9(8) COMP-4 SYNCHRONIZED
012700                                          VALUE 0.
012800
012900* DFH STRUC declarations
013000 COPY DFHAPIDC.
013100
013200 77 FUNCTION-HOLDER          PIC X(16).
013300
013400 77 CALL-LABEL                 PIC X(08).
013500 77 CALL-DETAILS               PIC X(80).
013600
013700 77 RESP                       PIC S9(8)  COMP-5 SYNC.
013800 77 RESP2                      PIC S9(8)  COMP-5 SYNC.
013900
014000* Channel browse stuff
014100 01 CHANNEL-BROWSE.
014200    03 BROWSE-TOKEN            PIC S9(8) COMP-4 SYNC.
014300    03 BROWSED-NAME            PIC X(16).
014400
014500*---------------------------------------------------------------*
014600* Constants
014700*---------------------------------------------------------------*
014800
014900 77 UNEXPECTED-RESP-ASCODE     PIC X(04) VALUE  SPACE.
015000
015100*---------------------------------------------------------------*
015200 LINKAGE SECTION.
015300
015400 01 DFHFUNCTION-IN.
015500    03 FILLER                   PIC X OCCURS 1 TO 54
015600                                DEPENDING ON DFHFUNCTION-IN-LEN.
015700
015800 01 GET-MAIN-AREA.
015900    03 FILLER                   PIC X OCCURS 1 TO 65536
016000                                      DEPENDING ON GET-MAIN-LEN.
016100
016200
016300*
016400 PROCEDURE DIVISION.
016500 MAIN-PROG SECTION.
016600
016700     Display MM 'MM Start ***'.
016800
016900     PERFORM GET-DFHFUNCTION-CONTAINER.
017000
017100     PERFORM BROWSE-ALL-CONTAINERS.
017200
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000543

```
C17300     IF FUNCTION-DRIVEN = PI-SEND-REQUEST
C17400     OR FUNCTION-DRIVEN = PI-RECEIVE-REQUEST
C17500     OR FUNCTION-DRIVEN = PI-PROCESS-REQUEST
C17600     THEN
C17700        PERFORM DEFINE-DEFRESPONSE
C17800     END-IF.
C17900
C18000     Display MM '**** End ****'.
C18100
C18200     EXEC CICS RETURN
C18300     END-EXEC.
C18400
C18500
C18600*****************************************************************
C18700*
C18800* Retrieve the contents of the DFHFUNCTION container
C18900*
C19000*****************************************************************
C19100 GET DFHFUNCTION CONTAINER.
C19200
C19300     MOVE PI-DFHFUNCTION TO CONTAINER-NAME.
C19400     PERFORM GET-NAMED-CONTAINER.
C19500
C19600     IF CONTAINER-FOUND = 'Y'
C19700     THEN
C19800        MOVE CONTAINER-LEN TO DFHFUNCTION-IN-LEN
C19900        SET DFHFUNCTION-IN-PTR TO GETDATA-PTR
C20000        SET ADDRESS OF DFHFUNCTION-IN TO DFHFUNCTION-IN-PTR
C20100        MOVE DFH-FUNCTION-IN TO FUNCTION-DRIVEN
C20200     END-IF.
C20300
C20400
C20500*****************************************************************
C20600*
C20700* Retrieve a named container
C20800*
C20900*****************************************************************
C21000 GET NAMED CONTAINER.
C21100
C21200     MOVE 'N' TO CONTAINER-FOUND
C21300* Get the length of the name container from the input pipe.
C21400     MOVE 'SNIFFCCL' TO CALL-LABEL.
C21500     EXEC CICS GET
C21600               CONTAINER(CONTAINER-NAME)
C21700               NODATA
C21800               FLENGTH(CONTAINER-LEN)
C21900               NOHANDLE
C22000               RESP(RESP)
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000544

```
C2210C              RESP2(RESP2)
C2220C     END-EXEC.
C2230C
C2240C* If the container was't found, do nothing   Else GETMAIN
C2250C* suitable storage and retrieve it for real.
C2260C     EVALUATE RESP
C2270C        WHEN DFHRESP(CONTAINERERR)
C2280C           DISPLAY NN  Container NOT FOUND'
C2290C        WHEN DFHRESP(NORMAL)
C2300C           MOVE 'Y' TO CONTAINER FOUND
C2310C
C2320C           DISPLAY NN  ********************************'
C2330C           DISPLAY NN  Container Name   : 'CONTAINER-NAME
C2340C           DISPLAY NN  Content length   : ' CONTAINER-LEN
C2350C
C2360C           IF CONTAINER-LEN > 0
C2370C           THEN
C2380C              MOVE 'SNDFROUT' TO CALL-LABEL
C2390C              MOVE CONTAINER-LEN TO GETMAIN-LEN
C2400C              EXEC CICS GETMAIN SET(GETMAIN-PTR)
C2410C                                FLENGTH(GETMAIN-LEN)
C2420C              END-EXEC
C2430C              SET ADDRESS OF GETMAIN-AREA TO GETMAIN-PTR
C2440C              EXEC CICS GET CONTAINER(CONTAINER-NAME)
C2450C                           INTO(GETMAIN-AREA)
C2460C                           FLENGTH(CONTAINER-LEN)
C2470C              END-EXEC
C2480C              DISPLAY NN 'Container content.
C2490C                     GETMAIN AREA
C2500C
C2510C           ELSE
C2520C              DISPLAY NN 'Container EMPTY'
C2530C           END-IF
C2540C        WHEN OTHER
C2550C           DISPLAY NN  Error on GET CONTAINER   CONTAINER-NAME
C2560C           PERFORM GENERIC-ABEND
C2570C     END-EVALUATE.
C2580C
C2590C
C2600C*****************************************************************
C2610C*
C2620C* BROWSE all containers on the current Channel
C2630C*
C2640C*****************************************************************
C2650C BROWSE-ALL-CONTAINERS.
C2660C
C2670C     DISPLAY NN 'Containers on channel: .ist starts.'
C2680C
C2690C* Start the Channel Browse
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000545

```
C27COC     MOVE 'SNIFFCC1' TO CALL-LABEL.
C27DOC     EXEC CICS STARTBROWSE
C27EOC               CONTAINER
C27FOC               BROWSETOKEN(BROWSE-TOKEN)
C27GOC               RESP(RESP)
C27HOC               RESP2(RESP2)
C27IOC     END-EXEC.
C27JOC
C27KOC* Browse the next container name
C27LOC     PERFORM UNTIL RESP NOT = DFHRESP(NORMAL)
C27MOC
C27NOC        MOVE 'SNIFFCC4' TO CALL-LABEL
C27OOC        EXEC CICS GETNEXT
C27POC                  CONTAINER(BROWSED-NAME)
C27QOC                  BROWSETOKEN(BROWSE-TOKEN)
C27ROC                  RESP(RESP)
C27SOC                  RESP2(RESP2)
C27TOC        END-EXEC
C27UOC
C27VOC        IF RESP = DFHRESP(NORMAL)
C27WOC        THEN
C27XOC           MOVE BROWSED-NAME TO CONTAINER-NAME
C27YOC           PERFORM GET-NAMED-CONTAINER
C27ZOC        END-IF
C280OC
C281OC     END-PERFORM.
C282OC
C283OC     IF RESP NOT = DFHRESP(END)
C284OC     THEN
C285OC        PERFORM GENERIC-ABEND
C286OC     END-IF.
C287OC
C288OC     DISPLAY MM 'Containers on channel: list ends'
C289OC
C290OC* End the Channel Browse
C291OC     MOVE 'SNIFFCC5' TO CALL-LABEL.
C292OC     EXEC CICS ENDBROWSE
C293OC               CONTAINER
C294OC               BROWSE-TOKEN(BROWSE-TOKEN)
C295OC               RESP(RESP)
C296OC               RESP2(RESP2)
C297OC     END-EXEC.
C298OC     IF RESP NOT = DFHRESP(NORMAL)
C299OC     THEN
C300OC        PERFORM GENERIC-ABEND
C301OC     END-IF.
C302OC
C303OC
C304OC*****************************************************************
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000546

```
C31900*                                                                *
C32000* Delete the DFHRESPONSE container                               *
C32100*                                                                *
C32200******************************************************************
C32300 DELETE-DFHRESPONSE.
C32400
C32500     MOVE 'SNIFFCOB' TO CALL-LABEL.
C32600     EXEC CICS ASSIGN
C32700               CHANNEL(CHANNEL-NAME)
C32800               RES-(RES-)
C32900               RESP2(RESP2)
C33000     END-EXEC.
C33100
C33200     IF CHANNEL-NAME =  'DFHWEC-V1' THEN
C33300        DISPLAY NN ' >> a SOAP header processing program.....'
C33400     ELSE
C33500        EXEC CICS DELETE
C33600                  CONTAINER(PI-DFHRESPONSE)
C33700                  RESP(RESP)
C33800                  RESP2(RESP2)
C33900        END-EXEC
C34000        EVALUATE RESP
C34100           WHEN DFHRESP(NORMAL)
C34200              DISPLAY NN PI-DFHRESPONSE ' container deleted'
C34300           WHEN OTHER
C34400              PERFORM GENERIC-ABEND
C34500        END-EVALUATE
C34600     END-IF.
C34700
C34800
C34900******************************************************************
C35000*
C35100* Generic abend
C35200*
C35300******************************************************************
C35400 GENERIC-ABEND.
C35500
C35600
C35700      DISPLAY NN '***** Unable to continue ***** '.
C35800      DISPLAY NN 'Unexpected RESP code from EXEC CICS call'.
C35900      DISPLAY NN 'Call label : ' CALL-LABEL.
C36000      DISPLAY NN 'RESP  : ' RESP.
C36100      DISPLAY NN 'RESP2 : ' RESP2.
C36200      IF CALL-DETAILS NOT = SPACES
C36300      THEN
C36400         DISPLAY NN  'Further details:'
C36500                   CALL-DETAILS
C36600      END-IF.
C36700
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000547

```
C3680C      EXEC CICS ABEND
C3690C         ABCODE(UNEXPECTED-RESP-ABCODE)
C3700C      END-EXEC.
```

Sample output from the SNIFFER program

Example A-5 shows sample output from the SNIFFER program. The output is sent to CICS transient data queue CESE

Example A-5 SNIFFER program - sample output

```
SNIFFER :  *** Start ***
SNIFFER :  ********************************
SNIFFER :  Container Name   - DFHEJNCTION
SNIFFER :  Content length   - 00000016
SNIFFER :  Container content: RECEIVE-REQUEST
SNIFFER :  Containers on channel: list starts.
SNIFFER :  >                              <
SNIFFER :  Container Name   . DFH WRAPPLEHFLIST
SNIFFER :  Content length   . 00000000
SNIFFER :  Container EMPTY
SNIFFER :  ********************************
SNIFFER :  Container Name   : DFHRESPONSE
SNIFFER :  Content length   - 00000000
SNIFFER :  Container EMPTY
SNIFFER :  >------------------------------<
SNIFFER :  Container Name   : DFHEJNCTION
SNIFFER :  Content length   . 00000010
SNIFFER :  Container content: RECEIVE-REQUEST
SNIFFER :  >------------------------------<
SNIFFER :  Container Name   : DFHWS-SOAPACTION
SNIFFER :  Content length   : 00000002
SNIFFER :  Container content: ""
SNIFFER :  ********************************
SNIFFER :  Container Name   - DFHWS-URI
SNIFFER :  Content length   : 00000075
SNIFFER :  Container content: /exampleApp/inquireString +
SNIFFER :  >                              <
SNIFFER :  Container Name   . DFHREQUEST
SNIFFER :  Content length   . 00000307
SNIFFER :  Container content: <soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmlns:soapenc="http://
schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Header/><soapenv
:Body><m528:DFHCXCMH xmlns:m528="http://www.DFHCXCMH_DFHWS2LS_Request.com"><m528
:ca_req_str id>QIINQS</m528:ca_request_id><m528:ca_return_code>0</m528:ca_retur
n_code><m529:ca_response_message>
</m528:ca_response_message><m528:ca_inquire_single><m528:ca_user_ref_req>10</m5
28:ca_item_ref_req><m528:file></m528:filler> </m528:filler>
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000548

```
<p628:ca_single_item><p628:ca_sngl_item_ref>0</p628:ca_sngl_item_ref><p628:ca_s
ngl_description>
</p628:ca_sngl_description><p628:ca_sngl_department>0</p628:ca_sngl_department>
<p628:ca_sngl_cost>0.0</p628:ca_sngl_cost><p628:in_sng_stock>0</p628:in_sng_s
tock><p628:on_sngl_order>0</p628:on_sngl_order></p628:ca_single_item></p628:ca
_inquire_single></p628:DFHCXCMD></soapenv:Body></soapenv:Envelope>
SNIFFER :  >                                 <
SNIFFER :  Container name   : DFH_SERVICEPLIST
SNIFFER :  Content length   : 00000000
SNIFFER :  Container EMPTY
SNIFFER :  <---------------------------------
SNIFFER :  Container name   : DFHWS-PIPELINE
SNIFFER :  Content length   : 00000008
SNIFFER :  Container content: EXPIPE01
SNIFFER :  >---------------------------------<
SNIFFER :  Container name   : DFHWS-JSFRIC
SNIFFER :  Content length   : 00000009
SNIFFER :  Container content: CICSSJ
SNIFFER :  <---------------------------------<
SNIFFER :  Container name   : DFHWS-TRANID
SNIFFER :  Content length   : 00000004
SNIFFER :  Container content: CPIH
SNIFFER :  >---------------------------------<
SNIFFER :  Container name   : DFHWS-WEBSERVICE
SNIFFER :  Content length   : 00000007
SNIFFER :  Container content: inquireSingle
SNIFFER :  >                                 <
SNIFFER :  Container name   : DFHWS-APPHANDLER
SNIFFER :  Content length   : 00000000
SNIFFER :  Container content: DFHPITP
SNIFFER :  Containers on channel: List ends
SNIFFER :  DFHRESPONSE      container deleted
SNIFFER :  **** End ****
CICSMSG1:  >---------------------------------<
CICSMSG1:  Container name: : DFHWS-WEBSERVICE
CICSMSG1:  Container content: inquireSingle
CICSMSG1:  ---------------------------------
CICSMSG1:  Container name: . DFHWS-TRANID
CICSMSG1:  Container content. CWQS
SNIFFER :  *** Start ***
SNIFFER :  <---------------------------------
SNIFFER :  Container name   : DFHFUNCTION
SNIFFER :  Content length   : 00000016
SNIFFER :  Container content: SEND-RESPONSE
SNIFFER :  Containers on channel: list starts.
SNIFFER :  >---------------------------------<
SNIFFER :  Container name   . DFHRESPONSE
SNIFFER :  Content length   . 00001047
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000549

```
SNIFFER :  Container content:
<SOAP-ENV:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/" xmln
s:soapenc="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsd="http://www.w3.o
rg/2001/XMLSchema" xmlns:xsi ="http://www.w3.org/2001/XMLSchema-instance" xmlns:SO
AP-ENV="http://schemas.xmlsoap.org/soap/envelope/"><SOAP-ENV:Body><DFHCICMN xmlns
:ns1="http://www.DFHCICMN.DFHJAVDP.Response.com"><ca_request_id>CITMQ</ca
_request_id><ca_return_code>0</ca_return_code><ca_response_message>RETURNED
ITEM. REF =0010
</ca_response_message><ca_inquire_single><ca_item_ref_req>10</ca_item_ref_req>
<filler1>    </filler1><filler2>
</filler2><ca_single_item><ca_sngl_item_ref>10</ca_sngl_item_ref><ca_sngl_descr
iption>Ball Pens Black2pc
</ca_sngl_description><ca_sngl_department>10</ca_sngl_department><ca_sngl_cost>
002.90</ca_sngl_cost><ca_sngl_qty_stock>0</in_sngl_stock><on_sngl_order>0</on_sngl
_order></ca_single_item></ca_inquire_single></DFHCICMN></SOAP-ENV:Body></
/SOAP-ENV:Envelope>
SNIFFER :  >                                 <
SNIFFER :  Container Name   : DFH-HANDLERLIST
SNIFFER :  Content length   : 00000000
SNIFFER :  Container EMPTY
SNIFFER :  <---------------------------------
SNIFFER :  Container Name   : DFHFUNCTION
SNIFFER :  Content length   : 00000016
SNIFFER :  Container content: SEND-RESPONSE
SNIFFER :  >---------------------------------<
SNIFFER :  Container Name   : DFHWS-SOAPACTION
SNIFFER :  Content length   : 00000002
SNIFFER :  Container content: ""
SNIFFER :  >---------------------------------<
SNIFFER :  Container Name   : DFHWS-URI
SNIFFER :  Content length   : 00000025
SNIFFER :  Container content: /exampleApp/inquireSingle
SNIFFER :  <---------------------------------
SNIFFER :  Container Name   : DFH-SERVICELIST
SNIFFER :  Content length   : 00000000
SNIFFER :  Container EMPTY
SNIFFER :  >                                 <
SNIFFER :  Container Name   : DFHWS-PIPELINE
SNIFFER :  Content length   : 00000009
SNIFFER :  Container content: SOAPIPEDI
SNIFFER :  <---------------------------------
SNIFFER :  Container Name   : DFHWS-USERID
SNIFFER :  Content length   : 00000008
SNIFFER :  Container content: CICSUSER
SNIFFER :  >---------------------------------<
SNIFFER :  Container Name   : DFHWS-TRANID
SNIFFER :  Content length   : 00000004
SNIFFER :  Container content: CWXN
SNIFFER :  >---------------------------------<
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000550

```
SNIFFER :  Container name   : DFHWS-WEBSERVICE
SNIFFER :  Content length   : 00000032
SNIFFER :  Container content: InquireSingle
SNIFFER :  ********************************
SNIFFER :  Container name   : DFHWS-APPHANDLER
SNIFFER :  Content length   : 00000009
SNIFFER :  Container content: DFHHTTP
SNIFFER :  ********************************
SNIFFER :  Container name   : DFHWS-OPERATION
SNIFFER :  Content length   : 00000009
SNIFFER :  Container content: DFHDORMH
SNIFFER :  Containers on channel: List ends
SNIFFER :  **** End ****
```

A.4 Sample XML parser program - MYPARSER

The MYPARSER program shown in Example A-6 is used by the CIWSSECH header processing program to parse the SOAP header in the DFHWS-DFHHEADER container. It uses the COBOL XML PARSE statement to parse through the SOAP WS-Security header that was received in the COMMAREA, extracting the user ID and password. This extracted data is then returned in the COMMAREA for the calling program to use.

Refer to the following publications for more information on XML parsing within COBOL programs:

- *Enterprise COBOL for z/OS V3R3 Programming Guide, SC27-1412*
- *Enterprise COBOL for z/OS V3R3 Language Reference, SC27-1408*

Example A-6 Sample XML parser program - MYPARSER

```
000100 PROCESS CICS
000200 IDENTIFICATION DIVISION.
000300 PROGRAM-ID.  MYPARSER
000400
000500 DATE COMPILED.
000600 ENVIRONMENT DIVISION.
000700 CONFIGURATION SECTION.
000800 SPECIAL-NAMES.
000900 DATA DIVISION.
001000 WORKING-STORAGE SECTION.
001100 01 WS-START.
001200    03 FILLER                    PIC X(44)
001300       VALUE '**** MYPARSER WORKING STORAGE STARTS HERE ****.
001400    03 RESP                      PIC S9(8) COMP-5 SYNC.
001500    03 RESP2                     PIC S9(8) COMP-5 SYNC.
001600    03 I                         PIC S9(4) COMP-5 SYNC.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000551

```
CC170C   03 J                          PIC S9(4) COMP-5 SYNC.
CC180C   03 K                          PIC S9(4) COMP-5 SYNC.
CC190C   03 L                          PIC S9(4) COMP-5 SYNC.
CC200C   03 WORKAREA                   PIC X(24).
CC210C*****************************************************
CC220C* Data items for use by parser
CC230C*****************************************************
CC240C   03 IN-ELEM                    PIC X(1) VALUE 'N'.
CC250C   03 IN-REF-REC                 PIC X(1) VALUE 'N'.
CC260C   03 PARSE-ERROR                PIC X(1) VALUE 'N'.
CC270C
CC280C* The XML document to parse
CC290C   01 XML-DOCUMENT PIC X(1024).
CC300C
CC310C   01 USERNAME-XML TAG-FOUND  PIC X VALUE 'N'.
CC320C
CC330C*****************************************************
CC340C* Externally referenced data areas
CC350C*****************************************************
CC360C LINKAGE SECTION.
CC370C 01  DFHCOMMAREA.
CC380C   02  DFHCOMMAREA-LN          PIC X(LC24).
CC390C   02  CA-RSP REDEFINES DFHCOMMAREA-LN.
CC400C     05   CA-USER-ID           PIC X(8).
CC410C     05   CA-PASSWORD          PIC X(8).
CC420C     05   FILLER               PIC X(1008).
CC430C
CC440C*****************************************************
CC450C* Main line code begins                             *
CC460C*****************************************************
CC470C PROCEDURE DIVISION.
CC480C MAIN-PROCESSING SECTION.
CC490C**************************
CC500C*    Validate the commarea
CC510C**************************
CC520C     PERFORM INIT-AND-VALIDATE.
CC530C*****************************************************
CC540C* Received a valid COMMAREA so invoke the XML parser
CC550C* Invoke the XML parser passing it the XML message
CC560C* What is the CICS command used to invoke the XML parser?
CC570C*****************************************************
CC580C     XML PARSE XML-DOCUMENT
CC590C        PROCESSING PROCEDURE XML-HANDLER
CC600C     END-XML.
CC610C
CC620C     PERFORM PERFORM-RETURN-RESPONSE.
CC630C
CC640C     EXEC CICS RETURN END-EXEC.
CC650C MAIN-PROCESSING-END. EXIT.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000552

```
CC66OC
CC67OC*****************************************************************
CC68OC* The following section is executed as a callback routine
CC69OC* out of the XML PARSE statement above.
CC70OC* What is the name of the register we need to check on the
CC71OC* EVALUATE statement?
CC72OC*****************************************************************
CC73OC XML-HANDLER SECTION.
CC74OC     EVALUATE XML-EVENT
CC75OC        WHEN 'START-OF-ELEMENT'
CC76OC*---------------------------------------
CC77OC* Check if we have an element of interest
CC78OC* i.e. Username and Password
CC79OC*---------------------------------------
CC80OC           IF XML-TEXT = 'wssp:Username'
CC81OC              MOVE 'Y' TO IN-ELEM USERNAME-XML-TAG-FOUND
CC82OC           ELSE
CC83OC              IF XML-TEXT = 'wssp:Password'
CC84OC                 MOVE 'Y' TO IN-ELEM PASS-XML-TAG
CC85OC              END-IF
CC86OC           END-IF
CC87OC        WHEN 'CONTENT-CHARACTERS'
CC88OC*---------------------------------------------
CC89OC* If we are in an element we are interested in,
CC90OC* then extract its value
CC91OC*---------------------------------------------
CC92OC           IF IN-ELEM = 'Y'
CC93OC              PERFORM EXTRACT-USER-ID
CC94OC           ELSE
CC95OC              IF PASS-REQ = 'Y'
CC96OC                 PERFORM EXTRACT-PASSWORD
CC97OC              END-IF
CC98OC           END-IF
CC99OC        WHEN 'END-OF-ELEMENT'
C100OC           CONTINUE
C101OC        WHEN 'START-OF-DOCUMENT'
C102OC           CONTINUE
C103OC        WHEN 'END-OF-DOCUMENT'
C104OC           CONTINUE
C105OC        WHEN 'VERSION-INFORMATION'
C106OC           CONTINUE
C107OC        WHEN 'ENCODING-DECLARATION'
C108OC           CONTINUE
C109OC        WHEN 'STANDALONE-DECLARATION'
C110OC           CONTINUE
C111OC        WHEN 'ATTRIBUTE-NAME'
C112OC           CONTINUE
C113OC        WHEN 'ATTRIBUTE-CHARACTERS'
C114OC           CONTINUE
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000553

```
           WHEN 'ATTRIBUTE-CHARACTER'
              CONTINUE
           WHEN 'START-OF-CDATA-SECTION'
              CONTINUE
           WHEN 'END-OF-CDATA-SECTION'
              CONTINUE
           WHEN 'CONTENT-CHARACTER'
              CONTINUE
           WHEN 'PROCESSING-INSTRUCTION-TARGET'
              CONTINUE
           WHEN 'PROCESSING-INSTRUCTION-DATA'
              CONTINUE
           WHEN 'COMMENT'
              CONTINUE
           WHEN 'EXCEPTION'
      *****************************************************************
      * Exception handling paragraph
      * XML code 52 means that the code page being used does not match
      * the one specified in the document. This is expected with non
      * EBCDIC data, as it has been translated to EBCDIC since we
      * received it.
      *****************************************************************
              IF XML-CODE = 52 MOVE ZERO TO XML-CODE
              ELSE
      ***************************************
      * An exception occurred during parsing
      ***************************************
                MOVE 'Y' TO PARSE-ERROR
              END-IF
           WHEN OTHER
              MOVE 'Y' TO PARSE-ERROR
        END-EVALUATE.
    XML-HANDLER-END. EXIT.
    XML01-SUBROUTINES SECTION.
      *****************************************************************
      * The parser returns everything between the <xmltag> and
      * </xmltag> tags, including all whitespace (SP, CR, LF, NL, HT)
      * these are removed before saving the xmltag content.
      *****************************************************************
    STRIP-EXTRA-WHITESPACE.
        MOVE ZERO TO I J.
        MOVE LENGTH OF XML-TEXT TO L.
        IF L > LENGTH OF WORKAREA
              MOVE LENGTH OF WORKAREA TO L.
        MOVE XML-TEXT TO WORKAREA(1 : L).
        INSPECT WORKAREA(1 : L) CONVERTING X'0D250A' TO SPACES.
        INSPECT WORKAREA(1 : L) TALLYING I FOR LEADING SPACES.
        SUBTRACT I FROM L.
        INSPECT WORKAREA(I + 1 : L) TALLYING J FOR ALL SPACES.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000554

```
C1640C     SUBTRACT J FROM L.
C1650C     IF L > LENGTH OF CA-USERID
C1660C           MOVE LENGTH OF CA-USERID TO L.
C1670C     MOVE WORKAREA(I + 1 : L) TO CA-USERID(1 : L)
C1680C     MOVE 'N' to CA-F.FM.
C1690C
C1700C EXTRACT-PASSWORD.
C1710C     MOVE ZERO TO L J.
C1720C     MOVE LENGTH OF XML-TEXT TO L.
C1730C     IF L > LENGTH OF WORKAREA
C1740C           MOVE LENGTH OF WORKAREA TO L.
C1750C     MOVE XML-TEXT TO WORKAREA(1 : L).
C1760C     INSPECT WORKAREA(1 : L) CONVERTING X'0E0F1505' TO SPACES.
C1770C     INSPECT WORKAREA(1 : L) TALLYING I FOR LEADING SPACES.
C1780C     SUBTRACT I FROM L.
C1790C     INSPECT WORKAREA(I + 1 : L) TALLYING J FOR ALL SPACES.
C1800C     SUBTRACT J FROM L.
C1810C     IF L > LENGTH OF CA-PASSWORD
C1820C           MOVE LENGTH OF CA-PASSWORD TO L.
C1830C     MOVE WORKAREA(I + 1 : L) TO CA-PASSWORD(1 : L).
C1840C     MOVE 'N' to CA-REF-REQ.
C1850C
C1860C*****************
C1870C* Return to caller
C1880C*****************
C1890C RETURN-RESPONSE.
C1900C     IF RESP-ERROR    *
C1910C*******************************************
C1920C*       Bad news, got an XML parser exception
C1930C*******************************************
C1940C        MOVE 'EXCEPTION' to CA-RSP
C1950C     ELSE
C1960C       IF USERNAME-XML-TAG-FOUND = 'N'
C1970C*************************************************
C1980C*          Ignore, it was not for us...
C1990C*          Bad news, did not find the expected XML tag
C2000C*************************************************
C2010C           MOVE 'NOT FOUND' to CA-RSP
C2020C       END IF
C2030C     END IF.
C2040C
C2050C*********************************************
C2060C* Initialise and Validate the COMMAREA
C2070C*********************************************
C2080C INIT-AND-VALIDATE.
C2090C*******************
C2100C*    Acceptable size ?
C2110C*******************
C2120C     IF EIBCALEN > 150
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000555

```
C2130C     THEN
C2140C***********
C2150C*      yes
C2160C***********
C2170C        MOVE DFHCOMMAREA-CN(1:EIBCALEN) to XML-DOCUMENT
C2180C        MOVE SPACES TO CA-USER-ID, CA-PASSWORD
C2190C     ELSE
C2200C**********
C2210C*      no
C2220C**********
C2230C        INITIALIZE DFHCOMMAREA(1:EIBCALEN)
C2240C        MOVE 'DAD COMMAREA LENGTH !' to DFHCOMMAREA
C2250C        EXEC CICS RETURN END-EXEC
C2260C     END-IF.
```

A.5 Sample header processing program - CIWSSECR

The CIWSSECR program in Example A-7 is used to insert a <wsse:Security> header into a SOAP message.

It determines, by obtaining the data from DFHFJNCTION, if it has been invoked in a SEND REQUEST phase. It also determines if it has been invoked for the correct destination, from the data in DFHWS-URI. It then obtains the user ID that the transaction is currently executing under and inserts this into the WS-Security header. This header is then written into the DFHHEADER container and control is returned to the calling program.

Example: A-7 Sample header processing program - CIWSSECR

```
CCC10C*****************************************************************
CCC20C IDENTIFICATION DIVISION.
CCC30C PROGRAM-ID.  CIWSSECR
CCC40C*****************************************************************
CCC50C*                                                               *
CCC60C* This program:-                                                *
CCC70C* 1  Checks for SEND-REQUEST function, else returns             *
CCC80C* 2  Checks for correct URI from DFHWS-URI                       *
CCC90C* 3. Obtains user ID from DFHWS-USERID container                *
CCD00C* 4. Puts completed WS-Security header into the                 *
CCD10C*    DFHHEADER container                                        *
CCD20C*****************************************************************
CCD30C
CCD40C AUTHOR.      CHERREGA  CL.
CCD50C DATE-COMPILED.
CCD60C ENVIRONMENT DIVISION.
CCD70C CONFIGURATION SECTION
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000556

```
CC180C SPECIAL-NAMES.
CC190C DATA DIVISION.
CC200C WORKING-STORAGE SECTION.
CC210C 01 WS-START.
CC220C   05 FILLER                  PIC X(44)
CC230C       VALUE 'DFHEISTG STARTS HERE ***'.
CC240C   05 HW                      PIC X(10) VALUE 'CONVERSE '.
CC250C   05 WS-RESP                 PIC S9(8) COMP-5 SYNC.
CC260C   05 WS-RESP2                PIC S9(8) COMP-5 SYNC.
CC270C   05 WS-HEAD-PTR             USAGE IS POINTER.
CC280C   05 WS-HEAD-LEN             PIC S9(8) COMP-4.
CC290C   05 WS-URI-PTR              USAGE IS POINTER.
CC300C   05 WS-URI-LEN              PIC S9(8) COMP-4
CC310C   05 WS-FUNC-LEN             PIC S9(8) COMP-4 VALUE 16.
CC320C   05 WS-PIPE-LEN             PIC S9(8) COMP-4 VALUE 9.
CC330C   05 WS-GETPATH-PTR          USAGE IS POINTER.
CC340C   05 WS-GETPATH-LEN          PIC S9(8) COMP-4.
CC350C   05 WS-FAULT-STRING         PIC X(48) value spaces.
CC360C   05 WS-FAULT-CODE           PIC S9(8) COMP-5.
CC370C   05 WS-SOAP-LEVEL           PIC S9(8) COMP-5.
CC380C     88 WS-SOAP-11  VALUE 1.
CC390C     88 WS-SOAP-12  VALUE 2.
CC400C     88 WS-NOT-SOAP VALUE 10.
CC410C   05 WS-FUNC-AREA            PIC X(16).
CC420C   05 WS-PIPE-AREA            PIC X(8).
CC430C   05 WS-XBUF-AUTH            PIC X(48)
CC440C              VALUE  'Not authorized to place order.'.
CC450C   05 WS-AUTH-FAIL-RC         PIC X(48)
CC460C              VALUE  'Authorization failed for order request.'.
CC470C   05 WS-WSSE-HEADER.
CC480C      05 WS-WSSE-01 PIC X(15) VALUE '<wsse:Security '.
CC490C      05 WS-WSSE-02 PIC X(26) VALUE 'soap-env:mustUnderstand="1"'.
CC500C      05 WS-WSSE-03 PIC X(25) VALUE ' xmlns:wsse="http://docs.o',
CC510C      05 WS-WSSE-04 PIC X(25) VALUE 'asis-open.org/wss/2004/01/',
CC520C      05 WS-WSSE-05 PIC X(75) VALUE 'oasis-200401-wss-wssecurit',
CC530C      05 WS-WSSE-06 PIC X(18) VALUE 'y-secext-1.0.xsd">'.
CC540C      05 WS-WSSE-07 PIC X(20) VALUE '<wsse:UsernameToken>'.
CC550C      05 WS-WSSE-08 PIC X(16) VALUE '<wsse:Username>'.
CC560C      05 WS-WSSE-09 PIC X(38) VALUE 'WEBUSZ    .
CC570C      05 WS-WSSE-10 PIC X(16) VALUE '</wsse:Username>'.
CC580C      05 WS-WSSE-11 PIC X(21) VALUE '</wsse:UsernameToken>'.
CC590C      05 WS-WSSE-12 PIC X(16) VALUE '</wsse:Security>'.
CC600C
CC610C 01 MYPARSER-COM-LEN           PIC S9(4) COMP-4.
CC620C 01 WS-PSW.
CC630C      05 WS-WSSE-A PIC X(33)
CC640C                        VALUE '<wsse:Password Type 'http://do'.
CC650C      05 WS-WSSE-B PIC X(33)
CC660C                        VALUE 'cs.oasis-open.org/wss/2004/01/'.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000557

```
006700      05 WS-WSSC-C PIC X(30)
006800                    VALUE 'wasis-2004Cl-wsa-username-token'.
006900      05 WS-WSSE-C PIC X(30)
007000                    VALUE 'n-profile-1 C#PasswordText'.
007100      05 WS-WSSF-P PIC X(8) VALUE 'PREDICKS '.
007200      05 WS-WSSF-P PIC X(16)  VALUE 'e/wsse:Password>'.
007300 01 DA-PARSER-WSP.
007400    03 DA USER LC             PIC X(0).
007500    03 DA PASSWORD            PIC X(0).
007600
007700 LINKAGE SECTION.
007800 01 WS-HEAD-AREA.
007900    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-HEAD-LEN.
008000
008100 01 WS-URI-AREA.
008200    02 FILLER    PIC X OCCURS 256 DEPENDING ON WS-URI-LEN.
008300
008400 01 WS-GETMAIN-AREA.
008500    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS GETMAIN LEN.
008600
008700*****************************************************************
008800* Main line code begins                                         *
008900*****************************************************************
009000 PROCEDURE DIVISION.
009100 MAIN-PROCESSING SECTION.
009200
009300     PERFORM GET-FILE.ONE.
009400
009500     PERFORM GET D-P-URL.
009600
009700     PERFORM GET-DFHWS-UR1.
009800
009900     EVALUATE WS-URI-AREA(1:22)
010000       WHEN '/ws/Quote?destination'
010100         IF WS-FUNC-AREA = 'SEND-REQUEST '
010200            PERFORM GET-USERID
010300            PERFORM GET-SOAP-HEADER
010400            PERFORM PUT SOAP HEADER
010500         END IF
010600       WHEN OTHER
010700         CONTINUE
010800     END-EVALUATE.
010900
011000     EXEC CICS RETURN END-EXEC.
011100
011200 MAIN-PROCESSING-END. EXIT.
011300
011400*****************************************************************
011500*     ***************************************     *
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000558

```
C11600*              SUBROUTINES FOLLOW                 *
C11700*     ************************************     *
C11800*************************************************
C11900
C12000*************************************************************
C12100* Retrieve the PIPELINE name from DFHWS-PIPELINE container
C12200*************************************************************
C12300 GET-PIPELINE.
C12400     EXEC CICS
C12500        GET CONTAINER('DFHWS-PIPELINE')
C12600        INTO(WS-PIPE-AREA)
C12700        FLENGTH(WS-PIPE-LEN)
C12800        RESP(WS-RESP)
C12900     END-EXEC.
C13000
C13100
C13200*************************************************************
C13300* Retrieve the function type from DFHFUNCTION container
C13400*************************************************************
C13500 GET-DFHFUNCTION.
C13600     EXEC CICS
C13700        GET CONTAINER('DFHFUNCTION')
C13800        INTO(WS-FUNC-AREA)
C13900        FLENGTH(WS-FUNC-LEN)
C14000        RESP(WS-RESP)
C14100     END-EXEC.
C14200
C14300*************************************************
C14400* Check for correct length. Create SOAP fault if
C14500* not correct. If for incoming request then we
C14600* do the work, else we exit immediately.
C14700*************************************************
C14800     IF WS-FUNC-LEN NOT = 15
C14900        MOVE WS-PATH-FAILED TO WS-FAULT-STRING
C15000        PERFORM FAULT-MESSAGE
C15100     END-IF.
C15200
C15300*************************************************************
C15400* Retrieve the URI from the DFHWS URI container
C15500*************************************************************
C15600 GET-DFHWS-URI.
C15700     EXEC CICS
C15800        GET CONTAINER('DFHWS-URI')
C15900        SET(WS-URI-PTR)
C16000        FLENGTH(WS-URI-LEN)
C16100        RESP(WS-RESP)
C16200     END-EXEC.
C16300
C16400*******************************************
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000559

```
C1630C* Copy the input container to our storage
C1640C*****************************************
C1650C     IF WS-URI-LEN > 0
C1660C        SET ADDRESS OF WS-URI-AREA TO WS-URI-PTR
C1690C        MOVE WS-URI-LEN TO WS-GETMAIN-LEN
C1700C
C1710C        EXEC CICS GETMAIN
C1720C            SET(WS-GETMAIN-PTR)
C1730C            FLENGTH(WS-GETMAIN-LEN)
C1740C        END-EXEC
C1750C
C1760C        SET ADDRESS OF WS-GETMAIN-AREA TO WS-GETMAIN-PTR
C1770C        MOVE WS-URI-AREA TO WS-GETMAIN-AREA
C1780C        SET WS-URI-PTR TO WS-GETMAIN-PTR
C1790C        SET ADDRESS OF WS-URI-AREA TO WS-URI-PTR
C1800C     ELSE
C1810C        MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C1820C        PERFORM FAULT-MESSAGE
C1830C     END-IF.
C1840C
C1850C*************************************
C1860C* Retrieve the <wsse:Security> header
C1870C*************************************
C1880C GET-SOAP-HEADER.
C1890C     EXEC CICS
C1900C        GET CONTAINER('DFHHEADER')
C1910C        SET(WS-HEAD-PTR)
C1920C        FLENGTH(WS-HEAD-LEN)
C1930C        RESP(WS-RESP)
C1940C     END-EXEC.
C1950C
C1960C**********************
C1970C* Retrieve the user ID
C1980C**********************
C1990C GET-USERID.
C2000C     EXEC CICS
C2010C        GET CONTAINER('DFHWS-USERID')
C2020C        INTO(WS-WSSE-09)
C2030C        FLENGTH(LENGTH OF WS-WSSE-09)
C2040C        RESP(WS-RESP)
C2050C     END-EXEC.
C2060C
C2070C*********************************
C2080C* Write the <wsse:Security> header
C2090C*********************************
C2100C PUT-SOAP-HEADER.
C2110C     EXEC CICS
C2120C        PUT CONTAINER('DFHHEADER')
C2130C        FROM(WS-WSSE-HEADER)
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000560

```
C21400        FLENGTH(LENGTH OF WS-WSSE-HEADER)
C21500        RESP(WS-RESP)
C21600     END-EXEC.
C21700     DISPLAY MM '================================'
C21800     DISPLAY MM 'Container Name: : DFHHEADER        '.
C21900     DISPLAY MM 'Content length: '    LENGTH OF WS-WSSE-HEADER.
C22000     DISPLAY MM 'Container contents: ' WS-WSSE-HEADER.
C22100     DISPLAY MM '                                   '.
C22200**
C22300
C22400*****************************************************************
C22500* The user ID has been verified. We will now set the contents of
C22600* DFHWS-USERID container with this value.This will cause the
C22700* business logic (back-end program/s) to be executed with this
C22800* user ID, using it's access rights.
C22900*****************************************************************
C23000 SET-USER-ID.
C23100     EXEC CICS
C23200         PUT CONTAINER( 'DFHWS-USERID'
C23300         FROM(CA-USER-ID)
C23400         FLENGTH(length of CA-USER-ID)
C23500         DATATYPE(DFHVALUE(CHAR))
C23600         RESP(WS-RESP)
C23700     END-EXEC.
C23800
C23900*****************************************************************
C24000* We detected that the ca_request_id field contains an invalid
C24100* request. This is a CICS-NI error.
C24200*****************************************************************
C24300 FAULT-MESSAGE-SECTION.
C24400************************
C24500* Generate a SOAP Fault
C24600************************
C24700     EXEC CICS
C24800         GET CONTAINER( 'DFHWS-SOAPLEVEL'
C24900         INTO(WS-SOAP-LEVEL)
C25000         FLENGTH(WS-HEAD-LEN)
C25100         RESP(WS-RESP)
C25200     END-EXEC.
C25300
C25400*************************************************
C25500* Perform the XML PARSE and verify the user ID
C25600*************************************************
C25700
C25800 PARSE-AND-VERIFY.
C25900*****************************************************************
C26000*  The SOAP body XML data can now be parsed by MYPARSER
C26100*    LINK to the XML parser, it will return the CA-USER-ID
C26200*                                          and CA PASSWORD
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000561

```
C2630C*                                          or   EXCEPTION'
C2640C*                                          or   NOT FOUND'
C2650C*                                          or   BAD COMMA'
C2660C*****************************************************************
C2670C     EXEC CICS  LINK PROGRAM('MYPARSER')
C2680C                    COMMAREA(WS-HEAD-AREA)
C2690C                    LENGTH(MYPARSER-COPLEN)
C2700C     END EXEC.
C2710C     IF WS HEAD AREA(1:9) = 'EXCEPTION' or
C2720C                            'NOT FOUND' or
C2730C                            'BAD COMMA'
C2740C*****************************************************
C2750C*  Error found during the XML PARSE program execution
C2760C*****************************************************
C2770C         MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C2780C         PERFORM FAULT-MESSAGE
C2790C     ELSE
C2800C         MOVE WS-HEAD-AREA to CA-PARSER-RSP
C2810C************************************
C2820C*  OK, data found, so do the VERIFY
C2830C************************************
C2840C         EXEC CICS VERIFY
C2850C            PASSWORD(CA-P455WORD)
C2860C            USERID(CA-USER-ID)
C2870C            RESP(WS-RESP)
C2880C         END-EXEC
C2890C*****************
C2900C*  Successful ?
C2910C***************
C2920C         IF WS RES- = DFHRESP(NORMAL)
C2930C            PERFORM SET-USER-ID
C2940C         ELSE
C2950C            MOVE WS-NOT-AUTH TO WS-FAULT-STRING
C2960C            PERFORM FAULT-MESSAGE
C2970C         END-IF
C2980C     END-IF.
C2990C*****************************************
C3000C* MOVE CORRECT VERSION OF FAULTCODE
C3010C*****************************************
C3020C     IF WS SOAP = 1 MOVE DFHVALUE(CLIENT) TO WS FAULT CODE
C3030C     ELSE
C3040C        MOVE DFHVALUE(SENDER) TO WS-FAULT-CODE
C3050C     END-IF
C3060C
C3070C     EXEC CICS SOAPFAULT CREATE
C3080C               FAULTSTRING(WS-FAULT-STRING)
C3090C               FAULTSTRLEN(LENGTH OF WS-FAULT-STRING)
C3100C               FAULTCODE(WS-FAULT-CODE)
C3110C     END EXEC.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000562

```
CC120C FAULT-MESSAGE-END. EXIT.
```

A.6 Sample header processing program - CIWSSECS

Program CIWSSECS in Example A-8 is similar to CIWSSECH (see Appendix A.2, "Sample header processing program - CIWSSECH" on page 505) except that no password verification is done.

Example. A-8 Sample header processing program CIWSSECS

```
CC010C  PROCESS CICS
CC020C IDENTIFICATION DIVISION.
CC030C PROGRAM-ID.  CIWSSECS
CC040C*****************************************************************
CC050C*                                                               *
CC060C* This program:                                                 *
CC070C* 1. Checks for RECEIVE REQUEST function, else returns         *
CC080C* 2. Obtains data from DFHHEADER container                      *
CC090C* 3  Uses the PYPARSER program to extract the CA-USER-ID       *
CC100C*    field from the XML message contained w i t h DFHHEADER     *
CC110C* 4  Puts CA-USER-ID  into DFHWS-USERID conta ner              *
CC120C*****************************************************************
CC130C
CC140C AUTHOR.      Grant Ward Ahlr.
CC150C DATE COMPILED.
CC200C ENVIRONMENT DIVISION.
CC210C CONFIGURATION SECTION.
CC220C SPECIAL-NAMES.
CC230C DATA DIVISION.
CC240C WORKING-STORAGE SECTION
CC250C 01 WS-START.
CC260C   05 FILLER                    PIC X(44)
CC270C       VALUE 'CIWSSECS WORKING STORAGE STARTS HERE '.
CC280C   05 WS-RESP                    PIC S9(8) COMP-5 SYNC.
CC290C   05 WS-RESP2                   PIC S9(8) COMP-5 SYNC.
CC300C   05 WS HEAD PTR                USAGE IS POINTER.
CC310C   05 WS-HEAD-LEN                PIC S9(8) COMP-4.
CC320C   05 WS-URI-PTR                 USAGE IS POINTER.
CC330C   05 WS-URI-LEN                 PIC S9(8) COMP-4.
CC340C   05 WS-FUNC-LEN                PIC S9(8) COMP-4 VALUE 16.
CC350C   05 WS-GETPATH-PTR             USAGE IS POINTER
CC360C   05 WS-GETPATH- FN             PIC S9(8) COMP-4.
CC370C   05 WS-FAULT-STRING            PIC X(48) va ue spaces.
CC380C   05 WS-FAULT-CODE              PIC S9(8) COMP-4.
CC390C   05 WS-SOAP-LEVEL              PIC S9(8) COMP-4.
CC400C     88 WS SOAP 11  VALUE 1.
CC410C     88 WS SOAP 12  VALUE 2.
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000563

```
CC42OC     88 WS-NOT-SCAP VALUE 10.
CC43OC   03 WS-FUNC-AREA             PIC X(16).
CC44OC   03 WS-NOT-AUTH              PIC X(40)
CC45OC               VALUE  'Not authorized to place order.'.
CC46OC   05 WS-AUTH-PAT PI           PIC X(40)
CC47OC               VALUE  'Authorization failed for order request.'.
CC48OC
CC49OC 01 MYPARSER COM-LEN            PIC S9(4) COMP 5.
CC5OOC 01 DA PARSER RSP.
CC51OC    03 DA USER ID             PIC X(8).
CC52OC    03 DA-PASSWORD            PIC X(8).
CC53OC
CC54OC LINKAGE SECTION.
CC55OC 01 WS-HEAD-AREA.
CC56OC    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-HEAD-LEN.
CC57OC
CC58OC 01 WS-URI-AREA.
CC59OC    02 FILLER    PIC X OCCURS 256 DEPENDING ON WS-URI-LEN.
CC6OOC
CC61OC 01 WS-GETMAIN-AREA.
CC62OC    02 FILLER    PIC X OCCURS 1024 DEPENDING ON WS-GETMAIN-LEN.
CC63OC
CC64OC*****************************************************************
CC65OC* Main line code beg *                                           *
CC66OC*****************************************************************
CC67OC PROCEDURE DIVISION.
CC68OC MAIN-PROCESSING SECTION.
CC69OC
CC7OOC**********************************
CC71OC*  Receive the SOA- Body namespace
CC72OC**********************************
CC73OC     PERFORM GET-DFHFUNCTION.
CC74OC
CC75OC**********************************
CC76OC*  Receive the SOAP Body namespace
CC77OC**********************************
CC78OC     PERFORM GET-DFHWS-URI.
CC79OC     IF WS URI AREA(1:4) NOT = 'wrq:'
CC8OOC        EXEC CICS RETURN END EXEC
CC81OC     END IF.
CC82OC
CC83OC**********************************
CC84OC*  Receive the SOAP Body namespace
CC85OC**********************************
CC86OC     PERFORM GET-SOAP-HEADER.
CC87OC
CC88OC*****************************************************************
CC89OC*  The SOAP body XML data can now be parsed by MYPARSER
CC9OOC*    LINK to the XML parser, it will return the DA USER ID
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000564

```
CC530C*                                          or   EXCEPTION'
CC540C*                                          or   NOT FOUND'
CC550C*                                          or   BAD COMMA'
CC560C*****************************************************************
CC570C     MOVE SPACES TO CA-PARSER-RSP
CC570C     EXEC CICS  link PROGRAM('MYPARSER')
CC580C                    COMMAREA(WS-HEAD-AREA)
CC590C                    LENGTH(MYPARSER_COPLEN)
C10000     END EXEC.
C10100     IF WS HEAD AREA(1:9) = 'EXCEPTION' or
C10200                           'NOT FOUND' or
C10300                           'BAD COMMA'
C10400*****************************************************
C10500*  Error found during the XML PARSE program execution
C10600*****************************************************
C10700         MOVE WS-ALTE-FAILED TO WS-FAULT-STRING
C10800         PERFORM FAULT-MESSAGE
C10900     ELSE
C11100*************************************
C11200*  OK, data found, so set the user ID
C11300*************************************
C11400         MOVE WS-HEAD-AREA to CA-PARSER-RSP
C12300         PERFORM SET-USER-ID
C12800     END-IF.
C12900*************************
C13000*  and it is the end, bye bye
C13100*************************
C13700     EXEC CICS RETURN END-EXEC.
C13800
C13900 MAIN PROCESSING END. EXIT.
C14000
C14100*****************************************************
C14200*     ************************************     *
C14300*             SUBROUTINES FOLLOW               *
C14400*     ************************************     *
C14500*****************************************************
C14600
C14700*************************************************************
C14800* Retrieve the function type from UPPERFUNCTION container
C14900*************************************************************
C15000 GET-DFHFUNCTION.
C15100     EXEC CICS
C15200        GET CONTAINER( DFHFUNCTION')
C15300        INTO(WS-FUNC-AREA)
C15400        FLENGTH(WS-FUNC-LEN)
C15500        RESP(WS-RESP)
C15600     END-EXEC.
C15700
C15800*****************************************************
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000565

```
C18400* Check for correct length. Create SOAP fault if
C18500* not correct. If for incoming request then we
C18600* do the work, else we exit immediately.
C18700*************************************************
C18800     IF WS-FUNC- Eh NOT - 15
C18900        MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C19000        PERFORM FAULT-MESSAGE
C19100     ELSE
C19200        IF WS FLAG AREA NOT = 'RECEIVE REQUEST'
C19300           EXEC CICS xE JPY END EXEC
C19400        END-IF
C19500     END-IF.
C19600
C19700*************************************************************
C19800* Retrieve the URI from the DFHWS-URI container
C19900*************************************************************
C20000 GET-DFHWS-URI.
C20100     EXEC CICS
C20200        GET CONTAINER( DFHWS JRI')
C20300        SET(WS-URI-PTR)
C20400        FLENGTH(WS-URI-LEN)
C20500        RESP(WS-RESP)
C20600     END-EXEC.
C20700
C20800*****************************************
C20900* Copy the input container to our storage
C21000*****************,***********************
C21100     IF WS-URI-.Eh > 0
C21200        SET ADDRESS Of WS JRI AREA IC WS URI PTr
C21300        MOVE WS URI LEN TO WS GETMAIN LEN
C21400
C21500        EXEC CICS GETMAIN
C21600            SET(WS-GETMAIN-PTR)
C21700            FLENGTH(WS-GETMAIN-LEN)
C21800        END-EXEC
C21900
C22000        SET ADDRESS Of WS-GETMAIN-AREA IC WS-GETMAIN-PTR
C22100        MOVE WS URI AREA TO WS GETMAIN AREA
C22200        SET WS URI PTr  O WS GETMAIN PTR
C22300        SET ADDRESS Of WS JRI AREA IC WS URI PTr
C22400     ELSE
C22500        MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C22600        PERFORM FAULT-MESSAGE
C22700        EXEC CICS RETURN END-EXEC
C22800     END-IF.
C22900
C23000*****************,***********************
C23100* Retrieve the <wsse:Security> header
C23200*************************************
```


Document Id: D3N4ZF3 4827 11E7 B66D A6EB9A1 66BC
Confidentiality: IBM



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000566

```
C20300 GET-SOAP-HEADER.
C20400     EXEC CICS
C20500        GET CONTAINER( DFHHEADER')
C20600        SET(WS-HEAD-PTR)
C20700        FLENGTH(WS-HEAD-LEN)
C20800        RESP(WS-RESP)
C20900     END-EXEC.
C21000
C21100*-------------------------------------
C21200* Copy the input container to our storage
C21300*-------------------------------------
C21400     IF WS-HEAD-LEN > 0
C21500        SET ADDRESS OF WS-HEAD-AREA TO WS-HEAD-PTR
C21600        MOVE WS-HEAD-LEN TO WS-GETMAIN-LEN OF PARSER-CONLEN
C21700
C21800        EXEC CICS GETMAIN
C21900            SET(WS-GETMAIN-PTR)
C22000            FLENGTH(WS-GETMAIN-LEN)
C22100        END EXEC
C22200
C22300        SET ADDRESS OF WS-GETMAIN-AREA TO WS-GETMAIN-PTR
C22400        MOVE WS-HEAD-AREA TO WS-GETMAIN-AREA
C22500        SET WS-HEAD-PTR TO WS-GETMAIN-PTR
C22600        SET ADDRESS OF WS-HEAD-AREA TO WS-HEAD-PTR
C22700     ELSE
C22800        MOVE WS-AUTH-FAILED TO WS-FAULT-STRING
C22900        PERFORM FAULT-MESSAGE
C23000        EXEC CICS RETURN END-EXEC
C23100     END-IF.
C23200
C23300*-------------------------------------------------------------
C23400* The user ID has been verified, we will now set the contents of
C23500* DFHWS-USERID container with this value.This will cause the
C23600* business logic (back-end program(s)) to be executed with this
C23700* user ID, using it's access rights.
C23800*-------------------------------------------------------------
C23900 SET-USER-ID.
C24000     EXEC CICS
C24100         PUT CONTAINER( DFHWS-USERID')
C24200         FROM(CA-USER-ID)
C24300         FLENGTH(length of CA-USER-ID)
C24400         DATATYPE(DFHVALUE(CHAR))
C24500         RESP(WS-RESP)
C24600     END-EXEC.
C24700
C24800*-------------------------------------------------------------
C24900* We determine that the ca_request_id field contains an invalid
C25000* request. This is a CLIENT error.
C25100*-------------------------------------------------------------
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000567

```
C2520C FAULT-MESSAGE SECTION.
C2530C**********************
C2540C* Generate a SOAP Fault
C2550C**********************
C2560C     EXEC CICS
C2570C          GET CONTAINER( DFHWS-SOAPLEVEL )
C2580C          INTO(WS-SOAP-LEVEL)
C2590C          FLENGTH(WS-READ-LEN)
C2600C          RESP(WS-RESP)
C2610C     END-EXEC.
C2620C
C2630C*************************************
C2640C* MOVE CORRECT VERSION OF FAULTCODE
C2650C*************************************
C2660C     IF WS-SOAP-11 MOVE DFHWSJF(CLIENT) TO WS-FAULT-CODE
C2670C     ELSE
C2680C        MOVE DFHWSJF(SENDER) TO WS-FAULT-CODE
C2690C     END-IF
C2700C
C2710C     EXEC CICS SOAPFAULT CREATE
C2720C               FAULTSTRING(WS-FAULT-STRING)
C2730C               FAULTSTRLEN(LENGTH OF WS-FAULT-STRING)
C2740C               FAULTCODE(WS-FAULT-CODE)
C2750C     END-EXEC.
C2760C FAULT-MESSAGE-END. EXIT.
```

A.7 Sample header processing program - WSATHND

The WSATHND C program shown in Example A-9 is used to monitor the exchange of registration and protocol service messages between WebSphere Application Server and CICS. It writes messages to the CESO transient data queue.

Example A-9 Sample header processing program - WSATHND

```
/*****************************************************************/
/*                                                               */
/* This program will perform the following functions when used as */
/* a  message handler in either the DFHWSATP pipeline or the     */
/* DFHWSATR pipeline:                                            */
/*                                                               */
/* 1) If MESSAGES ON is set to 1, it will write the following    */
/*    message:                                                   */
/*        WSAT: REACHED HANDLER   function                       */
/*    where function is the contents of the C-H-FUNCTION container */
/*    (RECEIVE REQUEST,SEND RESPONSE,SEND REQUEST,RECEIVE RESPONSE, */
/*    PROCESS-REQUEST,PC-RESPONSE,or HANDLER-CARRY). Recall that */
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000568

```
/*    When the function is NO-RESPONSE, then the handler is being   */
/*    invoked after processing a request, when there is no response */
/*    to be processed.                                              */
/*                                                                  */
/* 2) If the function is not NO-RESPONSE,then:                      */
/*    a) If FULL_MESSAGES_ON is set to 1, it will write the         */
/*       following message:                                         */
/*          MSG1: contents of the DFHREQUEST container              */
/*                                                                  */
/*    b) If MESSAGES_ON is set to 1, it will write the following    */
/*       message:                                                   */
/*          MSG1: ACTION: action                                    */
/*       where action is the character string aaaaaa which comes    */
/*       at the end of the following character strings found in the */
/*       DFHREQUEST container:                                      */
/*          http://schemas.xmlsoap.org/ws/2004/10/wscoor/aaaaaa     */
/*       and has one of the following values: RegisterResponse,     */
/*       Prepared,Committed, Register, Prepare, Commit, ReadOnly,   */
/*       Aborted, Abort, Rollback                                   */
/*                                                                  */
/* The messages are written to the CESO transient data queue which  */
/* is normally an extrapartition transient data queue with DDname   */
/* CESOUT                                                           */
/*                                                                  */
/********************************************************************/

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

/*******************************************************************/
/* The following flags can be set to either 0 or 1.                */
/*******************************************************************/
#define MESSAGES_ON        1
#define FULL_MESSAGES_ON   1

void check_response(char* containerr);

int main(char* argv[])
{
   /*****************/
   /*** VARIABLES ***/
   /*****************/

   /* Pointer to COMMAREA */
   char* commptr;

   /* Container names */
   char* cont_function = "DFHFUNCTION    \0";
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000569

```
char* cont_request  = "DFHREQUEST      \0";

/* Contents and length of DFHFUNCTION container */
char* function;
long int functionLen;

/* Contents and length of DFHREQUEST container */
char* message;
long int messageLen;

/* log message */
char* entryLogMsg;

char action[] = "               \0";

char* state1 =
     "http://schemas.xmlsoap.org/ws/2004/10/wscoor/RegisterResponse";
char* state2 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepared";
char* state3 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Committed";
char* state4 = "http://schemas.xmlsoap.org/ws/2004/10/wscoor/Register";
char* state5 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Prepare";
char* state6 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Commit";
char* state7 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/ReadOnly";
char* state8 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Aborted";
char* state9 = "http://schemas.xmlsoap.org/ws/2004/10/wsat/Abort";
char* state10= "http://schemas.xmlsoap.org/ws/2004/10/wsat/Rollback";

/***************/
/*** PROGRAM ***/
/***************/

EXEC CICS ADDRESS EIB(dfheiptr) CWAREA(commptr);

/*****************************************************************/
/* Store contents of DFHFUNCTION container in function variable */
/*****************************************************************/
EXEC CICS GET CONTAINER(cont_function) NODATA
          FLENGTH(functionLen);
check_response("WSATHND.getContainer.dfhfunction",

function = (char*) malloc(1 sizeof(char) * (1 + functionLen));

EXEC CICS GET CONTAINER(cont_function)
          INTO(function) FLENGTH(functionLen);
check_response("WSATHND.getContainer.dfhfunction");

*(function + functionLen)   '\0';

/*****************************************************/
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000570

```
/* Display message: WSAT: REACHED HANDLER - function  */
/*****************************************************/

entryLogMsg = (char*) malloc(sizeof(char*) *
                             (strlen("WSAT: REACHED HANDLER - ") +
                              strlen(function) + 1
                             )
                            );
strcpy(entryLogMsg, "WSAT - REACHED HANDLER   ");
strcat(entryLogMsg, function);
if (MESSAGES_ON == 1)
  printf("%s\n", entryLogMsg);

free(entryLogMsg);

/*****************************************************************/
/* Store contents of DFHREQUEST container in message variable */
/*****************************************************************/

if (!(strncmp(function, "NO-RESPONSE     ", 16) == 0))
{
    EXEC CICS GET CONTAINER(cont_request) NODATA
                 FLENGTH(messageLen);
    check_response("WSATHND.get.requestLen");

    message = (char*) malloc(sizeof(char*) * (messageLen+1));

    EXEC CICS GET CONTAINER(cont_request)
                 INTO(message) FLENGTH(messageLen),
    check_response("WSATHND.get.request");

    *(message+messageLen) = '\0';

    if (FULL_MESSAGES_ON == 1)
    {
      printf("WSAT:%s:\n", message);
    }

    /****************/
    /* ACTION STATE */
    /****************/


    if (strstr(message, state1) != NULL)
        strcpy(action, "RegisterResponse\0");
    else if (strstr(message, state2) != NULL)
        strcpy(action, "Prepared       \0");
    else if (strstr(message, state3) != NULL)
        strcpy(action, "Committed      \0");
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000571

```
        else if (strstr(message, state4) != NULL)
            strcpy(action, "Register        \0");
        else if (strstr(message, state5) != NULL)
            strcpy(action, "Prepare         \0");
        else if (strstr(message, state6) != NULL)
            strcpy(action, "Commit          \0");
        else if (strstr(message, state7) != NULL)
            strcpy(action, "ReadOnly        \0");
        else if (strstr(message, state8) != NULL)
            strcpy(action, "Aborted         \0");
        else if (strstr(message, state9) != NULL)
            strcpy(action, "Abort           \0");
        else if (strstr(message, state10) != NULL)
            strcpy(action, "Rollback        \0");

        /* (MESSAGES_ON == 1)
        printf("MSAT: ACTION=%s.\n", action);

        free(message);
    }

    EXEC CICS RETURN;
    check_response("mSATHND.rel,rm");

    return 0;
}


/* Check response of CICS commands - ensure no errors occur */
void check_response(char* identifier)
{
     if ((dfheiptr->eibresp + dfheiptr->eibresp2) != 0)
     {
         char* errStart = "ERROR: ";
         char* errEnd   = ". See stdout for responses";
         char* fullMsg;

         if (dfheiptr->eibresp == DFHRESP(NOTAUTH))
            printf("NOTAUTH Response found");
         if (dfheiptr->eibresp == DFHRESP(CCSIDERR))
            printf("CCSIDERR Response found");
         if (dfheiptr->eibresp == DFHRESP(CHANNELERR))
            printf("CHANNELERR Response found");
         if (dfheiptr->eibresp == DFHRESP(CONTAINERR))
            printf("CONTAINERR Response found");
         if (dfheiptr->eibresp == DFHRESP(INVREQ))
            printf("INVREQ Response found");
         if (dfheiptr->eibresp    DFHRESP(LENGERR))
            printf("LENGERR Response found");
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000572

```
        printf("Error: %s\n", identifier);
        printf("Error resp1: %d\n", dfheipt->eibresp);
        printf("Error resp2: %d\n", dfheipt->eibresp2);

        fullMsg = (char*) malloc(strlen(errStart) +
                                 strlen(identifier) +
                                 strlen(errEnd) + 1) *
                                 sizeof(char*));

        strcpy(fullMsg, errStart);
        strcat(fullMsg, identifier);
        strcat(fullMsg, errEnd);

        EXEC CICS SEND TEXT FROM(fullMsg)
                    LENGTH(strlen(fullMsg));

        free(fullMsg);

        exit;
    }
|
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000573

B

How the DES, AES, and HMAC algorithms work

Some readers may want or need a more detailed explanation of one or more of the following:

- How the DES algorithm works
- How the AES algorithm works
- How the HMAC algorithm of FIPS PUB 198 works

In this appendix we attempt to provide an overview of this information that does not require an understanding of advanced mathematics.

© Copyright IBM Corp. 2006. All rights reserved.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000574

B.1 How DES works

DES makes some use of permutations. A permutation of a set of elements is an arrangement of the elements of the set in some order. For example, there are six permutations of the set {1,2,3}: 123, 132, 213, 231, 312, and 321. The permutation 123 and the permutation 132 are considered to be different permutations because the order of the elements is different.

A sketch of the DES enciphering computation is given in Figure B-1. The 64 bits of the input block to be enciphered are first subjected to a permutation. In DES the permuted input has bit 58 of the input as its first bit, bit 50 of the input as its second bit, and so on as specified in FIPS Pub 46-3 with bit 7 of the input as its last bit. Figure B-1 shows the leftmost 32 bits of the permuted input as L_0, and the rightmost 32 bits of the permuted input as R_0. The permuted input block is then the input to a complex key-dependent computation described in "The cipher function f" on page 547. The output of that computation, called the preoutput, is then subjected to a permutation that is the inverse of the initial permutation.

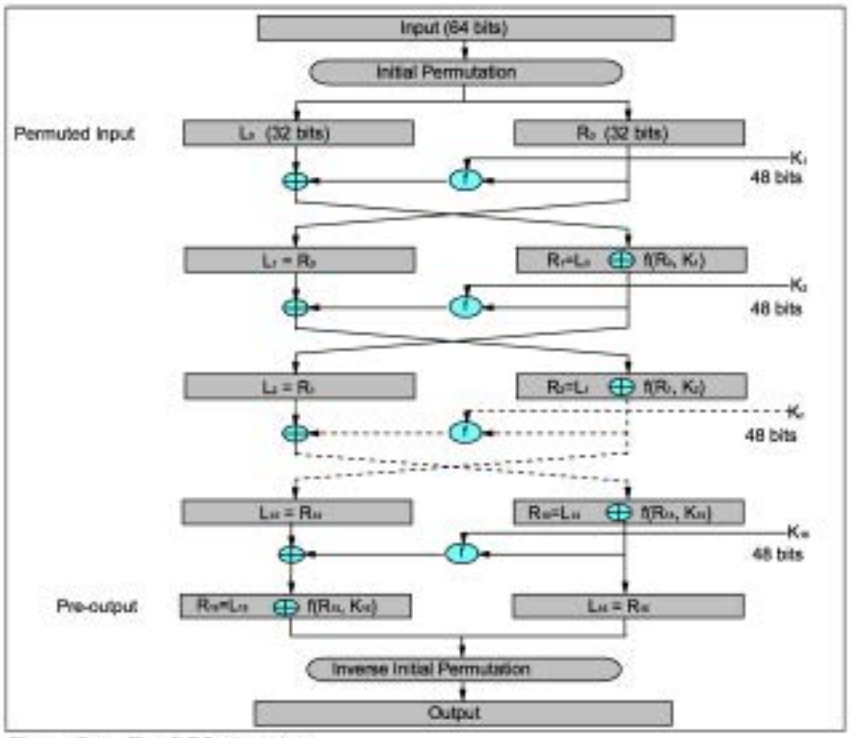


Figure B-1 The DES algorithm

Document Id: DD1942F0-662F-11EF-B95D-455B89A1668C
OnlineNotary.net
Page 575/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000575

The computation that uses the permuted input block as its input to produce the preoutput block consists, but for a final interchange of blocks, of 16 iterations of the calculation:

$$L_n = R_{n-1}$$

$$R_n = L_{n-1} \oplus f(R_{n-1}, K_n)$$

R_{n-1} represents the rightmost 32 bits of the output from the previous iteration. K_n represents a permuted selection of 48 bits chosen from the 64-bit key by the key schedule algorithm described in "Key schedule algorithm" on page 548. Thus the cipher function f operates on two blocks, one of 32 bits and one of 48 bits; it produces a block of 32 bits, which is then XORed with L_{n-1} to produce R_n. (XOR is a bit-by-bit exclusive OR operation in which $1 \oplus 1 = 0, 1 \oplus 0 = 1, 0 \oplus 1 = 1, 0 \oplus 0 = 0$. XOR is equivalent to a bit-by-bit addition modulo 2.) The preoutput block is then $R_{16}L_{16}$.

The cipher function f

A sketch of the calculation of f(R,K) is given in Figure B-2.

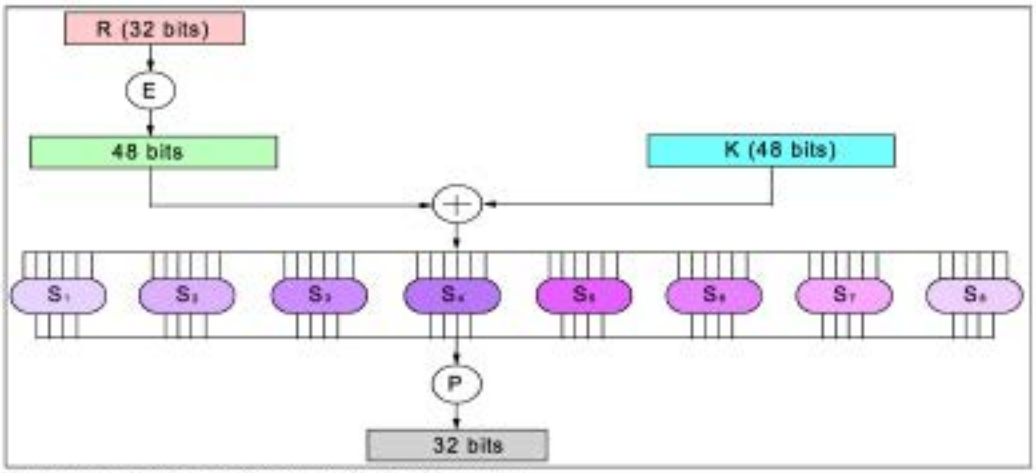


Figure B-2 The cipher function f of the DES algorithm

E denotes a function that takes a block of 32 bits as input and produces a block of 48 bits as output. The first three bits of E(R) are the bits in positions 32, 1, and 2 of R, the last 2 bits of E(R) are the bits in position 32 and 1 of R, and the remaining bits of E(R) are chosen from the bits of R according to a table specified in FIPS Pub 46-3.

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 576/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000576

Each of the unique selection functions $S_1, S_2 \ldots S_8$ takes a 6-bit block of input and yields a 4-bit block as output as specified in FIPS Pub 46-3.

The 8 blocks of 4 bits each are consolidated into a single block of 32 bits, which forms the input to the permutation function P. P yields a 32-bit output from a 32-bit input by permuting the bits of the input block, again as specified in FIPS Pub 46-3. The output is then the output of the cipher function f for the inputs R and K.

Key schedule algorithm

Figure B-3 shows a sketch of the key schedule calculation of the DES algorithm. The bits of the key are numbered 1 through 64. The bits of C_0 are respectively bits 57,49, 41, 33, 25 . .. 44 and 36 of the key. The bits of D_0 are respectively bits 63, 55, 47, 39, 31, . . 12, and 4 of the key. Permuted Choice 1 does not select any of the eight parity bits of the key for either C_0 or D_0.

C_1 and D_1 are obtained from C_0 and D_0 respectively by one left shift. C_2 and D_2 are obtained from C_1 and D_1 respectively by one left shift. C_8 and D_8 are obtained from C_7 and D_7 by one left shift, and C_{16} and D_{16} are obtained from C_{15} and D_{15} by one left shift. In all other cases C_n and D_n are obtained from C_{n-1} and D_{n-1} by two left shifts. In all cases, by *one left shift* is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3, ... 28, 1.

Permuted Choice 2 selects the 48 bits of K_n from the 56 bits of C_nD_n as specified by a table in FIPS Pub 46-3. The first bit of K_n is the 14th bit of C_nD_n, the second bit is the 17th, and so on with the 47th bit of K_n being the 29th bit of C_nD_n, and the 48th bit of K_n being the 32nd bit of C_nD_n.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000577

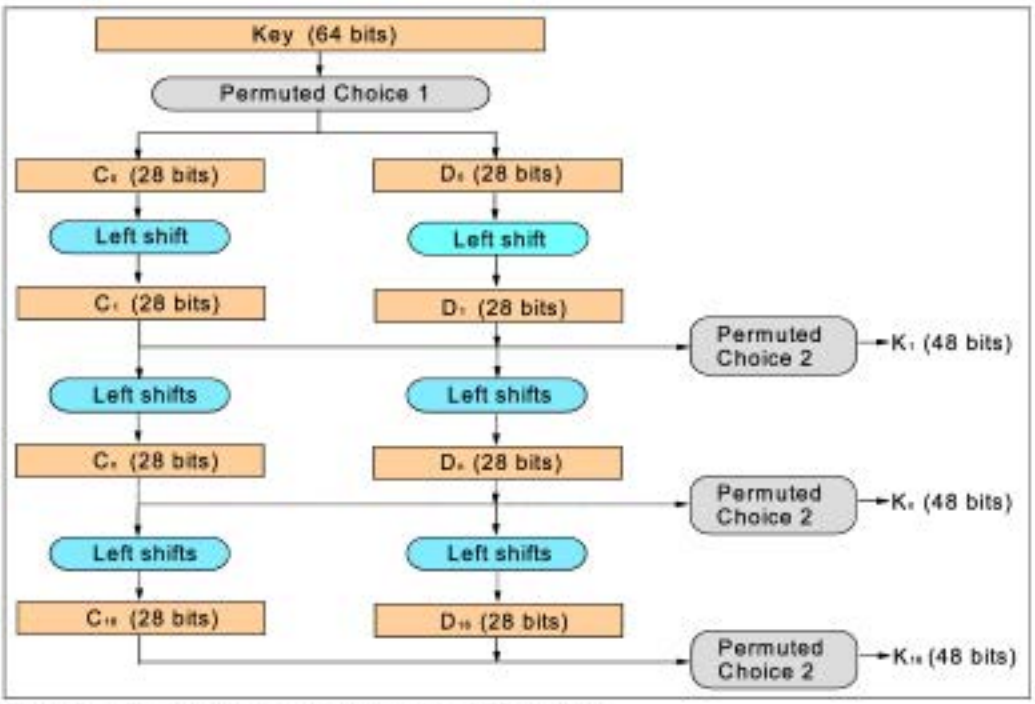


Figure B-3 The key schedule calculation of the DES algorithm

B.2 How AES works

The AES algorithm requires an initial set of 4 words of key data, and each of the Nr rounds requires 4 words of key data. Thus the algorithm requires 4 + 4*Nr = 4(Nr+1) words of key data. Consequently the algorithm specifies a Key Expansion routine that accepts the cipher key as input and uses it to generate a key schedule containing 4(Nr+1) words as output. The first Nk words of the key schedule are the first Nk words of the cipher key. For example, if we specify the following 128-bit cipher key:

2b 7e 15 16 28 ae d2 a6 ab f7 15 88 09 cf 4f 3c

then the Key Expansion routine generates a key schedule containing 4(10 + 1) = 44 words, of which the first 4 words are as follows:

w_0 = 2b7e1516 w_1 = 28aed2a6 w_2 = abf71588 w_3 = 09cf4f3c

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineHistory.net
Page 578/6

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000578

Example B-1 shows pseudo code that describes how AES encrypts data. The pseudo code uses four variables.

- The variable *in* is an array of 16 bytes that contains the 128 bits to be encrypted.
- The variable *out* is an array of 16 bytes that contains the data after it has been encrypted.
- The variable *w* is an array of 4*(Nr + 1) words: it contains the key schedule that the Key Expansion routine generated.
- The variable *state* is a rectangular array having 4 rows and 4 columns of bytes that represent intermediate encryption results.

Example B-1 Pseudo code for AES encryption

```
Cipher( byte in[16], byte out[16], word w[4*(Nr+1)])
begin
    byte state[4,4]

    state = in

    AddRoundKey(state, w[0,3] )

    for round = 1 step 1 to Nr-1
       SubBytes(state)
       ShiftRows(state)
       MixColumns(state)
       AddRoundKey(state, w[4*round, 4*round+3])
    end for

    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[4*Nr, 4*Nr+3])

    out = state
end
```


Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000579

The input array, *in*, is copied into the *state* array according to the scheme shown in Example B-2

Example B-2 state = in

```
s[0,0] = in[0]  s[0,1] = in[4]  s[0,2] = in[8]  s[0,3] = in[12]
s[1,0] = in[1]  s[1,1] = in[5]  s[1,2] = in[9]  s[1,3] = in[13]
s[2,0] = in[2]  s[2,1] = in[6]  s[2,2] = in[10] s[2,3] = in[14]
s[3,0] = in[3]  s[3,1] = in[7]  s[3,2] = in[11] s[3,3] = in[15]
```

For example, if the data to be encrypted is:

32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34

then the initial value of the *state* array would be as shown in Table B-1.

Table B-1 state = in

32	**88**	**31**	**e0**
43	5a	31	37
f6	30	98	07
a8	8d	a2	34

Note that the first column of the *state* array contains the first 32-bit word of the input, the second column of the *state* array contains the second word of the input, and so forth. The *state* can hence be interpreted as a one-dimensional array of 32-bit words (columns): w0, w1, w2, w3.

The encryption operations are then conducted on this *state* array. After an initial Round Key addition, the *state* array is transformed by implementing a round function 10, 12, or 14 times (depending on the key length), with the final round differing slightly from the first Nr-1 rounds because it does not include the MixColumns() transformation. The final state is then copied to the output.

The individual transformations - SubBytes(), ShiftRows(), MixColumns(), and AddRoundKey() - which make up the round function that processes the state array are described in the following sections.

SubBytes() transformation

The SubBytes() transformation is a substitution that operates independently on each byte of the *state* array using the substitution table set forth in Figure 7 of FIPS Pub 197. The substitution table contains 16 rows and 16 columns. If, for example, the $s_{1,1}$ element of the *state* array contained the byte 53, then the substitution value would be found at the intersection of the row with index '5' and

Document Id: [illegible]
Confidentiality: [illegible]
Page 000580

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000580

the column with index '3' in the substitution table. This would result in 53 being replaced with ed.

The creators of AES used complex mathematics to create the substitution table in such a way that it would have an inverse substitution that could be used in a decryption operation.

ShiftRows() transformation

In the ShiftRows() transformation the bytes in the second row of the *state* array are cyclically shifted left by one column. The bytes in the third row are cyclically shifted left by two columns. The bytes in the last row are cyclically shifted left by three columns. The first row is not shifted. See Figure B-4.



Figure B-4 ShiftRows() cyclically shifts the last three rows in the state array

MixColumns() transformation

The MixColumns() transformation, shown in Figure B-5, replaces the four bytes in column c of the state array, namely $\{s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}\}$, with four new bytes $\{s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}\}$ whose values are computed according to the following equations:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$
$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$
$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c})$$
$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

In these equations $a \oplus b$ represents the XOR operation, which is equivalent to addition modulo 2, and $x \bullet y$ represents the multiplication of two elements in the finite field GF(2^8). It is beyond the scope of this book to explain the mathematics of finite fields. However, we will note that the creators of AES chose the coefficients in the preceding equations to have certain specific values in the finite

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 581/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000581

field GF(2^8) so that the MixColumns() transformation would have an inverse transformation that could be used in a decryption operation.

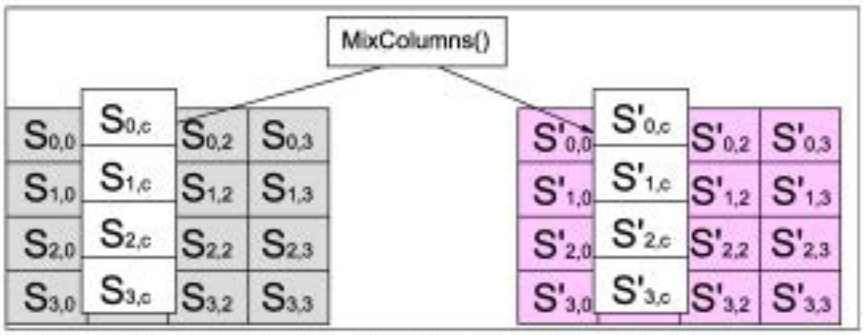


Figure B-5 MixColumns() operates on the state array column-by-column

AddRoundKey() transformation

Figure B-6 shows that the AddRoundKey() transformation XORs each column of the *state* array with a word from the key schedule. In Figure B-6 I is equal to four times the number of the round. The AddRoundKey() transformation performs an XOR operation between the first column of the state array and word w_i from the key schedule in order to create the first column of the new *state* array. Likewise, it performs an XOR operation between the second column of the *state* array and word w_{i+1} from the key schedule to create the second column of the new state array. Likewise, column three is XORed with w_{i+2}, and column four is XORed with w_{i+3}.

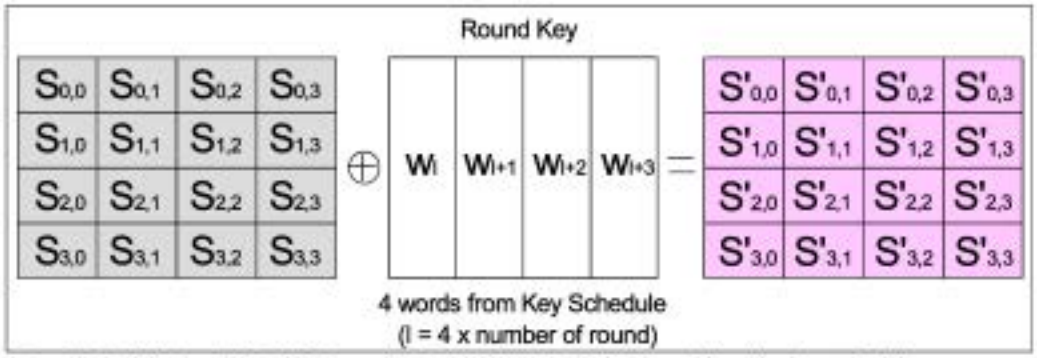


Figure B-6 AddRoundKey() XORs each column of the state with a word from the key schedule

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net
Page 582/6
Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000582

B.3 How the HMAC algorithm of FIPS PUB 198 works

FIPS PUB 198 describes an HMAC using the parameters shown in Table B-2.

Table B-2 Parameters for the HMAC algorithm

Parameter	Description
B	Block size (in bytes) of the input to the hash function
H	Hash function (such as SHA-1)
ipad	Inner pad, the byte x'36' repeated B times
K	Secret key shared between the originator and the intended receiver(s)
K_0	The key K after any necessary pre-processing to form a key of length B
L	Block size (in bytes) of the output of the hash function
opad	Outer pad, the byte x'5C' repeated B times
l	The number of bytes of MAC
text	The data on which the HMAC is calculated; *text* does not include the padded key. The length of text is n bits

A well-known practice with MACs is to truncate their output; that is, the length of the MAC used is less than the length L of the output of the hash function. When a truncated HMAC is used, the l leftmost bytes of the HMAC computation are used as the MAC. The output length, l, should be no less than four bytes, so $4 \le l \le L$.

To compute a MAC over the data *text* using the HMAC function, FIPS PUB 198 specifies the performance of the following operation:

$$MAC(text, K) = HMAC(K, text) = H((K_0 \oplus opad) \| H((K_0 \oplus ipad) \| text))$$

where $a \oplus b$ indicates an Exclusive-OR operation and $\|$ indicates a concatenation operation. Table B-3 describes the step-by-step process in the HMAC algorithm.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000583

Table B-3 The HMAC algorithm

Step	Description of step
1	If the length of $K = B$: set $K_0 = K$. Go to step 4.
2	If the length of $K > B$: hash K to obtain a string L bytes long, then append $(B-L)$ zeros to create string K_0 which is B bytes long That is, $K_0 = H(K) \| 00...00$. Then go to step 4.
3	If the length of $K < B$: append zeros to the end of K to create a string K_0 which is B bytes long. (For example, if K is 20 bytes in length and $B = 64$, then K will be appended with 44 zero bytes X'00').
4	Exclusive-OR K_0 with *ipad* to produce the string $K_0 \oplus ipad$. This string has length B.
5	Append the stream of data 'text' to the string resulting from step 4: $(K_0 \oplus ipad) \| text$
6	Apply the hash function H to the stream generated in step 5: $H((K_0 \oplus ipad) \| text)$
7	Exclusive-OR K_0 with *opad* to produce the string $K_0 \oplus opad$. This string has length B.
8	Append the result from step 6 to step 7: $(K_0 \oplus opad) \| H((K_0 \oplus ipad) \| text)$
9	Apply the hash function H to the result from step 8: $H((K_0 \oplus opad) \| H((K_0 \oplus ipad) \| text))$
10	Select the leftmost I bytes of the result of step 9 as the MAC

The successful verification of a MAC does not completely guarantee that the accompanying message is authentic: there is a chance that a source with no knowledge of the key can present a purported MAC on the plaintext message that will pass the verification procedure. For example, an arbitrary purported MAC of I bits on an arbitrary plaintext message may be successfully verified with an expected probability of $(1/2)^I$. Therefore, in general, if the MAC is truncated, then its length I should be chosen as large as is practical, with at least half as many bits as the output block size, L.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000584

Document Id: [illegible]
Client Security: [illegible]



Page 556(5)

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000585

Abbreviations and acronyms

2PC	Two-phase commit
AAT	Application Assembly Tool
AOR	Application-Owning Region
API	Application Programming Interface
APPC	Advanced Program-to-Program Communications
ASCII	American Standard Code for Information Interchange
B2B	Business-to-business
CCI	Common Client Interface
CEX2	Crypto Express2
CICS	Customer Information Control System
CICS TG	CICS Transaction Gateway
CORBA	Common Object Request Broker Architecture
CPACF	CP Assist for Cryptographic Functions
DPL	Distributed Program Link
DSA	Digital Signature Algorithm
EAR	Enterprise Application Archive
EBCDIC	Extended Binary Coded Decimal Interchange Code
ECI	External Call Interface
EIS	Enterprise Information Systems
EJB	Enterprise JavaBeans™
EPI	External Presentation Interface
ESI	External Security Interface
ESM	External Security Manager
EXCI	External CICS Interface
FTP	File Transfer Protocol
GTF	Generalized Trace Facility
GUI	Graphical user interface
HFS	Hierarchical File System
HTML	Hypertext Transfer Protocol
HTTP	Hypertext Markup Language
IBM	International Business Machines
ICSF	Integrated Cryptographic Service Facility
IDE	Integrated development environment
IRC	inter-region communication
ISC	Inter-System Communication
ITSO	International Technical Support Organization
J2C	J2EE Connector Architecture (also known as JCA)
J2EE	Java 2 Enterprise Edition
JAAS	Java Authentication and Authorization Service
JAR	Java Archive
JAX-RPC	Java API for XML-based RPC
JCA	J2EE Connector Architecture (also known as J2C)
JDBC	Java Database Connectivity
JDK™	Java Developer's Kit
JKS	Java Keystore
JMS	Java Messaging Service
JNDI	Java Naming and Directory Interface™
JNI	Java Native Interface
JSP	JavaServer Page

© Copyright IBM Corp. 2006. All rights reserved.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000586

JSSE	Java Secure Sockets Extension
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
LPAR	Logical Partition
LTPA	Lightweight Third-Party Authentication
LUW	Logical Unit of Work
MRO	Multi Region Operation
PEM	Password Expiration Management
PKDS	Private Key Data Set
QoS	Quality of service
RAD	Rational Application Developer
RAR	Resource Adapter Archive
RPC	Remote procedure call
RRMS	Recoverable Resource Management Services
RRS	Resource Recovery Services
RSA	Rivest, Shamir, and Adelman (encryption)
SAF	System Authorization Facility
SAX	Simple API for XML
SDK	Software Development Kit
SNA	Systems Network Architecture
SOAP	Simple Object Access Protocol (also known as Service-Oriented Architecture Protocol)
SQL	Structured query language
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security
TPV	Tivoli® Performance Viewer
UDDI	Universal Description, Discovery and Integration
UOW	Unit of Work
URI	Uniform resource identifier
URL	Uniform resource locator
URN	Uniform resource name
VSAM	Virtual Storage Access Method
WMQ	WebSphere MQ
WS-A	Web Services - Addressing
WS-AT	Web Services - Atomic Transaction
WS-BA	Web Services - Business Activity
WS-C	Web Services - Coordination
WSDL	Web Services Description Language
WSGW	Web Services Gateway
WS-I	Web Services Interoperability
WS-Security	Web Services - Security
XMI	XML metadata interchange
XML	Extensible Markup Language
XSD	XML Schema Definition

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000587

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see "How to get IBM Redbooks" on page 560. Note that some of the documents referenced here may be available in softcopy only.

- *Application Development for CICS Web Services*, SG24-7126
- *WebSphere Version 6 Web Services Handbook Development and Deployment*, SG24-6461
- *Web Services Handbook for WebSphere Application Server 6.1*, SG247257
- *Patterns: Service-Oriented Architecture and Web Services*, SG24-6303
- *WebSphere MQ in a z/OS Parallel Sysplex Environment*, SG24-6864
- *Websphere MQ Security in an Enterprise Environment*, SG24-6814

Other publications

These publications are also relevant as further information sources:

- *Enterprise COBOL for z/OS V3R3 Programming Guide*, SC27-1412
- *Enterprise COBOL for z/OS V3R3 Language Reference*, SC27-1408
- *CICS Web Services Guide V3.1*, SC34-6458
- *CICS TS V3.1 RACF Security Guide*, SC34-6249
- *CICS TS V3.1 Application Programming Reference*, SC34-6434
- ICSF Overview, SA22-7519
- RACF Command Reference, SA22-7687
- *CICS Transaction Server for z/OS RACF Security Guide*, SC34-6454
- *Your Web Services Atomic Transaction operations: Beginner's guide to classic transactions, data recovery, and mapping to WS-Atomic Transactions*. IBM developerWorks (September 2, 2004)

© Copyright IBM Corp. 2006. All rights reserved.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000588

- *WebSphere MQ for z/OS System Setup Guide V6.0*, SC34-6583
- *WebSphere MQ - Transport for SOAP*, SC34-6651
- *Websphere MQ Security*, SC34- 6588

Online resources

These Web sites and URLs are also relevant as further information sources:

- CICS library

  `http://www.ibm.com/software/htp/cics/library/`

- WebSphere Application Server library

  `http://www.ibm.com/software/webservers/appserv/was/library/index.html`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Document Id: DD1942F0-662F-11EF-B95D-455B90A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000589

Index

A
aborted 413
AddRoundKey 553
Advanced Encryption Standard (AES) 150
AES algorithm
  how it works 545
alert protocol 197
APARs
  PK10849 439
  PK16509 437
  PK20393 113
application sample 76
Application Server Toolkit V6.1 (AST) 305
asymmetric cryptography 150
ATM
  application 385
atomic transaction 381
AtomicClient 439–440, 445, 461
auditing 150
authentication 152, 231
  basic 302
authorization 152
Automatic Teller Machine (ATM) 382
availability 126
  configuring for 101

B
basic 217
basic authentication 217, 302
  sample 292, 308
bean-managed transaction 450
block cipher 154
bus
  see service integration bus

C
C language 34
C++ 34
catalog application 122, 439, 461
catalog manager 146
catalog manager application 67
  configuration transaction ECFG 97, 122, 465
  testing 90, 100, 124, 145
CatalogEar 97
CatalogAtomic.ear 450
CatalogSer.ear 280
CatalogSec_WS-Security.ear 265, 346, 370
CatalogSec_WS-Security_HTTPS.ear 267
CEDA 119, 443
CEMT 120
certificate 274, 319
certificate revocation 184
certification paths 182
channels 59
CICS 295
  certificate 322
  COBOL programs 65
  configuring 311
  encryptor 221
  Execution Diagnostic Facility (EDF) 125
  security 214
  service provider 57
    preparation 36
    using HTTP 76–77
    using WMQ 116
  service requester 41
    local optimization 45
    preparation 42
    timeout considerations 96
    using HTTP 94–95
    using WMQ 121
  trace 105
  Web services assistant 34, 62
    DFHLS2WS 63
    DFHWS2LS 64
  Web services support 33
    catalog manager application 67
    development tools 61
    high availability 101, 126
    overview 34
    problem determination 104
    resources 47
    resources checklist 55
  WS-Security Component 279
CICS default user ID 215
CICS region user ID 215

© Copyright IBM Corp. 2006. All rights reserved.

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000590

CICSplex 102
CICS-to-CICS 112
cipher 154
  suite 193
Cipher Block Chaining (CBC) 155
cipher suite 290
CIWSMSGH 80, 93, 118, 500
CIWSSECH 265–269, 505
CIWSSECR 267, 527
CIWSSECS 269, 303, 534
COBOL 34
code page 74, 77
code page conversion 77
commit 393, 413
commitand 413
Communications Manager 102
compilation process 410
confidentiality 152
connection 79
connector 439
container-managed transaction 450
containers 59
  context container 59
  control container 59
  header container 60
  user container 61
CoordinationContext 400
  elements 468
    Address 469
    CoordinationType 469
    Expires 468
    Identifier 468
    ReferenceProperties 469
    RegistrationService 469
  example 467
  sample created by CICS 402
  sample created by WebSphere 403
coordinator 406, 436
CORBA 8
core standards 8
CP Assist for Cryptographic Functions (CPACF) 209, 321
CPIH 41, 93
CPIL 116
credentials 311
Crypto Express2 (CEX2) feature 209, 321
cryptographic hardware 209
cryptography 151, 153
  hardware 321
  public key 160
  secret key 153
  symmetric 154
cryptosystem 151

D
daisy chain 436
Data 155
Data Encryption Algorithm (DEA) 155
Data Encryption Standard (DES) 156
DB2 tasks 439
decrypting 153
default user ID, CICS 215
DES algorithm
  how it works 545
development tools 61
DFHSRING 275, 299
DFHLS2WS 34, 63
DFHPIHS 429
DFHPISNT, message handler module 429, 440
DFHRESPONSE container 82
DFHRSLRI, URIMAP resource 426
DFHWS2LS 34, 64
DFHWSATH, header processing program 420, 427–429, 440
DFHWSATP, provider pipeline 421–422
DFHWSATR, requester pipeline 421
  PIPELINE definition 424
DFHWSATX, message handler 421, 424
Diffie-Hellman 160
digital signature 153, 168, 330
Digital Signature Algorithm (DSA) 171
dispatchOrder.ear 96
DispatchOrderAtomic.ear 402
distributed routing 102–103
DPL subset 87
dynamic program routing 102, 104
dynamic response 47
dynamic routing 104

E
EAR file 99
eavesdropping 215
ECFG 97, 122, 465
Electronic Codebook (ECB) 155
electronic data interchange (EDI) 4
Elliptic Curve Digital Signature Algorithm (ECDSA) 173

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000591

encrypting 153
encryption algorithm 195
endpoint
  listener 131
  reference 391
enterprise bean 450
error 330
  handling 16
error handling 197
ExampleAppClient.ear 87
ExampleAppDispatchOrder 88
ExampleAppDispatchOrder.ear 89, 462
Execution Diagnostic Facility (EDF) 125
Extensible Markup Language
  see XML

F
FTP 9, 276

G
gateway
  instance 131
    configuring 140
  service 131, 135
    configuring 139, 141
    invoking 147
    publishing 145
    testing 145

H
handler program 499
handshake protocol 193, 198
hardware 209
  cryptography 321
hash algorithm 193
hashing 153, 164
heuristic decision 416
HFS file system 74
high availability 101, 126
  with HTTP 101
  with WMQ 126
HMAC algorithm
  how it works 545
HTTP 7, 36, 73, 215
hub configuration 426

I
IBM Java SDK 277
ibm-webservices-bnd.xmi 259, 304
ibm-webservicesclient-bnd.xmi 259, 304
ibm-webservicesclient-ext.xmi 259, 304
ibm-webservices-ext.xmi 259, 304
identification 152
identity assertion 244
iKeyMan 277
IMS 65
inbound service 131
in-doubt window 415
Integrated Cryptographic Service Facility (ICSF)
210
integrity 152
intermediary 242
intermediate gateway
  identity assertion 244
Internet 3
iterated block cipher 151

J
J2EE
  application 146
  connector 439
J2EE Connector Architecture (JCA) 496
Java SDK 277
Java Transaction API (JTA) 432
JAX-RPC
  handler 130, 131
JCA 439
JDBC 439
JDBC provider 459
JMS 9, 439
JVM 69

K
key agreement protocol 189
key exchange algorithm 193
key exchange protocol 189
key pair
  generation 319
keyring 274
  creating 274
key schedule algorithm 548
keystore 277

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000592

L
local optimization 45

M
master secret 201
mediator 130–131
medium 222
message authentication codes (MAC) 166
message handler 35, 90, 116, 296
  introduction 57
  SOAP 57
  writing 90
message handler program 300
message information headers 394
mixcolumns 552
mustUnderstand 14, 265, 311, 316, 351, 375
MVS 102
MYPARSER 268, 522

N
namespace 12
Namespaces 12
National Institute of Standards and Technology (NIST) 155
non repudiation 153
non terminal message 415
notification message 414

O
one way message 414
outbound service 131

P
Parallel Sysplex 102
participant 417, 436
performance 304
PIPELINE 49
  attributes
    CONFIGFILE 50
    SHELF 50
    WSDIR 49
  defining 83, 95, 119
pipeline 35, 49, 60, 118, 265, 301
  configuration file 51
    customizing 90
pipeline alias transaction 48, 57, 93
PLT 34
Port definition 29
prepared 413
presumed abort 383
presumed trust 244
problem 316
  determination 104
protocol transformation 132
public key cryptography 150
  cryptography
    public key 151

R
RACF 325
  keyring 299
Rational Application Developer 265, 280
readonly 413
recoverable 383
Redbooks Web site 560
  Contact us xxii
reference
  parameter 393
  property 393
Register
  request 404
  response 406
remote procedure call (RPC) 17
replay 413
request consumer 240
request generator 240, 359
RequestStream 431
response consumer 241, 338
response generator 241
rollback 383, 413
RPC 17
  style 33
RSA algorithm 167

S
sample application 76, 287, 409
scalability 101
secret 152
secret key cryptography 153
  cryptography
    secret key 151
Secure Sockets Layer (SSL) 215
security
  auditing 153
  authentication 152

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000593

authorization 152
confidentiality 152
exposures 215
HTTP 217
identification 152
integrity 152
non repudiation 153
prefixing 300
setup 299
SIT parameters 300
SOAP message 217, 257
see WS-Security
token 302
transport-level 217
comparison with SOAP message security 241
WMQ 224, 293
testing 291
self-signed 319
service
broker 5
gateway 135
see gateway service
inbound 131
mediator 130
outbound 131
provider 5
requestor 5
service integration bus 129–130, 139
configuration 132
securing CICS Web services 242
service provider 76
service provider mode 303
service requester
installing 87
timeout considerations 124
service requester setup 304
service-oriented approach 4
service-oriented architecture 3–4
characteristics 5
shillrowe 552
signature
processing 347
SIT parameters 76, 300
DFLTUSER 215
ENCRYPTION 219
KEYRING 219, 281, 300
MAXSSLTCBS 219
SCC 251, 300
SECPRFX 251, 300
SSLCACHE 220
SSLDELAY 219
TCPIP 76
XTRAN 251, 300
XUSER 251, 300
SMTP 9
SNIFFER 136, 440, 511
sample output 519
SOA
see service-oriented architecture
SOAP 6–9
1.1 specification 9
1.2 specification 10
binding 30
body 15, 469
outbound data conversion 41
communication styles 17
document 17
RPC 17
encodings 17
literal 17
SOAP encoding 17
envelope 12–13
fault 16
fault messages 271, 314, 324
headers 14, 468
intermediary 14
introduction 12
message 296, 330
message encryption 357
message handler 35, 57
message security 217, 225, 241
messaging mode 17
MustUnderstand 15
namespaces 12
security headers 284
server 7
Simple Object Access Protocol 9
validation in CICS 50
SOAPFAULT
commands
ADD 56
CREATE 56
generating a SOAP fault in CICS 271
SOCKETCLOSE 79
spoofing 215
SSL/TLS 273
CICS support for 219

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000594

creating a key ring 271
enabling CICS support 279, 281
enabling WebSphere support 276
testing 282
with WMQ 226
standards
JAX RPC 133
JCA 456
JTA 432
SOAP 9
UDDI 10
WS-Addressing 381
WS-Atomic Transaction 381
WS-Coordination 381
WSDL 10
WS-Security 11
WS-Transactions 11
XML 8
static response 47
Station-to-Station (STS) protocol 197
strong 222
subbyles 551
subkey 154
surrogate 214
surrogate access 313
switch profiles 324
symmetric cryptography 154
Sysplex Distributor 102, 128

T
tampering 215
tbsCertificate 175
TCP/IP 417
load balancing 102
TCPIP.SIT parameter 79
TCPIPSERVICE 55, 78, 261
attributes
AUTHENTICATE 216
PORTNUMBER 79
PROTOCOL 79
SSL 220
terminal message 415
testing 90, 100, 124, 270, 282, 300, 461, 489
throughput 204
token 302
trading
CICS 105
transactional scope 365
transactions
atomic 389, 401
enabling CICS support 415
business activity 401
classic 382, 383
two phase commit 383
transport 111, 241
security 217
Transport Layer Security (TLS) 191
trust token 244
trusted 160
two phase commit (2PC) 383
presumed abort 384
walkthrough 395
with WS-AT 412

U
UDDI 8
Unicode 77
Universal Description, Discovery, and Integration (UDDI) 10
URI 123
URIMAP 47, 120, 135, 252, 302
attributes
USERID 223
defining 85, 135, 253
for serving HFS file 135
setting the userID 252
UsernameToken 303
UsernameTokenGenerator 263, 309

V
verification 172

W
Waico 382
weak 222
Web 287
Web service 53
Web Service Description Language (WSDL) 8
Web services 3, 287
assistant 62
binding file 55
definition 7
introduction 6
properties 7
Web Services Description Language

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000595

see WSD_
Web Services Gateway 130
Web Services Interoperability Organization
see WS-
Web services versus service-oriented architectures 6
Web technologies 3
WEBSERVICE 59
attributes
VALIDATION 53
WSBIND 53
WSDLFILE 53
dynamically installing 65
WebSphere 239
certificate 324
key store 326
MQ 111
service requester
deploying 86, 90–99
using HTTP 87, 98
WebSphere Application Server 298
administrative console 68
connection pooling 89
WebSphere Developer for zSeries 65
WebSphere MQ
see WMQ
WMQ 112
configuration 114
defining queues 115
trigger process 116
shared queues 127–128
WS Binding 260
WS Binding page 263, 309
WS Extension page 261, 307
WS-Addressing 381, 390
CICS Registration Service endpoint 396
endpoint reference 391
specification 390
WSATHND 444, 509
WS-Atomic Transaction (WS-AT) 36, 381, 387, 408
Completion protocol 410
daisy chain test scenario 462
testing 489
enabling CICS support 420, 421
enabling WebSphere support 432, 459
resynchronization processing 415
simple test scenario 441
testing 461
specification 400
Two-Phase Commit protocol 411
Durable 2PC 411
Volatile 2PC 411
typical scenario 439
walkthrough 387, 414
WS-Business Activity specification 400
unsupported by CICS 401
WS-Coordination 381
Activation service 390
Coordination service 398
Protocol services 396
Registration service 398
specification 397
walkthrough 406
WSDL 10, 18
binding 19, 27
bindings 30
definition 24
document 19
anatomy 19
message 19, 25
namespaces 23
operation 19, 26
port 19, 29
port type 19, 26
service 19
service definition 29
SOAP binding 30
type 19
types 24
Web Services Description Language 10
WSGW 130
WS-I
basic profile 10
introduction 6
WS-Security 226, 257
CICS support 229
configuration
files 265, 311–313, 350, 372
configuring CICS 265
example 231, 255, 311, 372
message authentication 231
specifications 226
test scenario 270, 313, 350, 372
typical scenario 228
WebSphere client configuration 260, 306
WebSphere configuration files 258, 304
with WMQ 234
WS-Security Component 229

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000596

X
X.509 230, 286
XML 7, 9
XML encryption 357
XML Services for the Enterprise (XSE) 65

Z
z/OS 102, 210
  Communications Manager 102
  Parallel Sysplex 102

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000597

Document Id: DD1942F0-662F-11EF-B95D-455899A1668C
OnlineNotary.net

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000598

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000599

Implementing CICS Web Services



Redbooks

Configuring and securing Web services in CICS Transaction Server

Connecting CICS to a service integration bus

Enabling atomic Web services

The Web services support in CICS Transaction Server Version 3.1 enables your CICS programs to be Web service providers or requesters. CICS supports a number of specifications including SOAP Version 1.1 and Version 1.2, and Web services distributed transactions (WS-Atomic Transaction).

This IBM redbook will help you configure the CICS Web services support for both HTTP and WebSphere MQ based solutions. We show how Web services can be used to integrate J2EE applications running in WebSphere Application Server with COBOL programs running in CICS.

It begins with an overview of Web services standards and the Web services support provided by CICS TS V3.1. Complete details for configuring CICS Web services using both HTTP and WebSphere MQ are provided next, along with the steps for using Web services to connect to CICS from a service integration bus. The book then shows how CICS Web services can be secured using a combination of Web Services Security (WS-Security) and transport-level security mechanisms such as SSL/TLS. Finally, it demonstrates how atomic Web services transactions can be configured to allow WebSphere and CICS resource updates to be synchronized.

This book concentrates on implementation specifics such as security, transactions, and availability. The companion redbook *Developing CICS Web Services* (SG24-7126) presents detailed information on developing CICS Web services.

SG24-7206-01 ISBN 0738494534

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

Document Id: DD1942F0-662F-11EF-B95D-455B80A1668C
OnlineNotary.net



Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000601

JURAT ATTACHMENT

A notary public or other officer completing this certificate verifies only the identity of the individual who signed the document to which this certificate is attached, and not the truthfulness, accuracy, or validity of that document.

STATE OF Texas }

COUNTY OF Tarrant }

The foregoing instrument was subscribed and sworn before me this date of 08/29/2024, by Nathaniel Frank-White

This notarial act was an online notarization.



(Notary Seal)

Notary's Signature _____

Registration No.: 133645929

Commission Expiration Date: March 15, 2026

Amazon.com, Inc., Amazon.com Services LLC - Ex. 1007, Page 000602