SOAP for CICS feature

User's Guide

SC34-6315-00

SOAP for CICS feature

User's Guide

SC34-6315-00

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 53.

First edition (September, 2003)

This edition applies to the SOAP for CICS feature, for CICS Transaction Server for z/OS Version 2 program number 5697-E93, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

At the back of this publication is a page entitled "Sending your comments to IBM". If you wish to send comments by mail, please address them to:

User Technologies Department
Mail Point 095
IBM United Kingdom Laboratories
Hursley Park
WINCHESTER
Hampshire
SO21 2JN
United Kingdom

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Chapter 1. The SOAP for CICS feature

The SOAP for CICS feature provides support for the Simple Object Access Protocol (SOAP) version 1.1 in CICS® Transaction Server Version 2.

The SOAP for CICS feature enables existing or new CICS applications, written in any supported programming language, to communicate outside of the CICS environment using the Simple Object Access Protocol (SOAP). The feature supports inbound and outbound SOAP requests. In other words:

- A SOAP client application (or service requester) can call a CICS application program
- A CICS application can call a SOAP server application (or service provider)

A user-written program known as a *message adapter* provides a mapping between XML and the CICS communication area (COMMAREA) used by an application program. WebSphere® Studio Enterprise Developer (WSED) provides a tool to generate converter routines from COBOL copybooks, which can perform the mapping. You can also develop mappings using the XML parsing function provided in Enterprise COBOL V3 and Enterprise PL/I V3.

The feature provides two message transports:

Hypertext Transfer Protocol (HTTP)
The HTTP transport uses CICS Web support, and includes support for the Secure Sockets layer (SSL)

WebSphere MQ
The WebSphere MQ transport uses the CICS adapter supplied with WebSphere MQ for z/OS®

The SOAP for CICS feature enables a user-written application layer to map the XML-based SOAP message into a COMMAREA, thus enabling access to COMMAREA-based applications using SOAP messages.

The feature supports the Simple Object Access Protocol (SOAP) 1.1 specification (`http://www.w3.org/TR/SOAP/`). It implements a SOAP processing node which, by default, understands no SOAP headers (see section 4.2 of the specification). The default implementation:

- ignores optional headers
- rejects requests that contain mandatory headers

However, you can provide your own code to process SOAP headers.

Where to find more information

To use this feature successfully, you need some knowledge of the following CICS functions:

Business Transaction Services (BTS)
User-written code in the feature uses a small subset of the extensive repertoire of BTS commands. They are described in "BTS commands" on page 5. For general information about BTS, see the *CICS Business Transaction Services* manual.

You should also be familiar with the following:

Extended Markup Language (XML)
The formal specification of XML is published at `http://www.w3.org/XML`.

Simple Object Access Protocol (SOAP)
The formal specification of SOAP is published at `http://www.w3.org/TR/SOAP`.

If you plan to build Web Services, you will need some knowledge of the following:

Web Services Description Language (WSDL)
The formal specification of WSDL is published at `http://www.w3.org/TR/SOAP`.

As well as the formal specifications, there is a large body of information on the internet, and many printed books that provide detailed information on these subjects.

If you plan to use the HTTP transport, you should be familiar with the following:

CICS Web support
For more information, see the *CICS Internet Guide*.

Hypertext Transfer Protocol (HTTP)
The formal specification of HTTP is published at `http://www.w3.org/Protocols.`

If you plan to use the WebSphere MQ transport, you should be familiar with using the CICS adapter function of WebSphere MQ. For more information, see *WebSphere MQ for z/OS Concepts and Planning Guide*.

What's in this feature?

Load modules

Library SCAVLOAD contains the following load modules:

DFHWSABE
The abend handler program for the service provider pipeline

DFHWSAMX
The default application mapper program

DFHWSDSH
The service provider dispatcher program for the HTTP transport

DFHWSDSQ
The service provider dispatcher program for the WebSphere MQ transport

DFHWSHDX
The default SOAP message handler program

DFHWSPMI
The service provider pipeline manager

DFHWSPMO
The service requester pipeline manager

DFHWSRDO
Resource definitions used by the feature

DFHWSRT
The outbound SOAP router program

DFHWSSH
The SOAP protocol handler

DFHWSTIH
The HTTP transport program for the service provider pipeline

DFHWSTIQ
The WebSphere MQ transport program for the service provider pipeline

DFHWSTOH
The HTTP transport program for the service requester pipeline

DFHWSTOQ
The WebSphere MQ transport program for the service requester pipeline

DFH0SBM
Mapset used by the DFH$WSSB sample program

DFH$WSAP
Back-end program of the Web Service sample application

DFH$WSBT
Document template used by the DFH$WSSB sample program

DFH$WSDC
Front-end program of the Web Service sample application

DFH$WSSB
The main program of the DFH$WSSB sample

DFH$WSSS
The sample service provider program

DFH$WSXL
The sample XML tracing program

Sample programs

Library SCAVSAMP contains the following sample source code:

DFH$WSAP
COBOL source for the back-end program of the Web Service sample application

DFH$WSBT
Assembler source for the document template used by the DFH$WSSB sample program

DFH$WSDC
COBOL source for the front-end program of the Web Service sample application

DFH$WSDL
Web Service Definition Language (WSDL) for the Web Service sample application

DFH$WSSB
COBOL source for the sample service requester program

DFH$WSSS
COBOL source for the sample service provider program

DFH$WSXL
COBOL source for the sample XML tracing program

DFHWSAMX
COBOL source for the default application mapper program

DFHWSCSD
CSD utility program commands for defining resources used by the feature

DFHWSCNO
COBOL copy book containing constants used by DFH$WSSB and DFH$WSSS

DFHWSHDX
COBOL source for the default SOAP message handler program

DFH0SBMP
Assembler source for the mapset for the DFH$WSSB sample program

The sample programs are described in Chapter 9, "The sample applications," on page 45

Installing and using the SOAP for CICS feature

1. To install the SOAP for CICS feature, follow the directions in the *SOAP for CICS feature Program Directory*.
2. To use the feature in your CICS system, include the load library in the DFHRPL concatenation for your CICS job.

Migrating from the SOAP for CICS SupportPac

If you have previously used the SOAP for CICS SupportPac™ CA1M, you can use both the SupportPac and this feature in the same region.

If you want to adapt programs and operating procedures that you have developed for use with the SupportPac with this feature, you should be aware that many of the program names used by the feature are not the same as those in the SupportPac. You should make the following changes.

- Rename any user handlers you have written.
  1. Rename the SOAP message handler program. Its name in the SupportPac is WSHANDLE; its name in the SOAP for CICS feature is DFHWSHDX.
  2. Rename the application mapper program. Its name in the SupportPac is WSAPPMAP; its name in the SOAP for CICS feature is DFHWSAMX.
- If you use the HTTP transport, change any references to the HTTP dispatcher program. Its name in the SupportPac is WSSOAPHT; its name in the SOAP for CICS feature is DFHWSDSH. For example, if you use the default analyzer, the URI of the target program is now:

  `http://host:port/CICS/CWBA/DFHWSDSH/program`

  where *program* is the name of the target program.
- If you use the MQ transport, change any references to the transport transaction. The old name is WSMQ; the new name is CWSQ.
- If you use the MQ transport, change any references to the MQ dispatcher program. The old name is WSSOAPMQ; the new name is DFHWSDSQ. For example, the definition of transaction CWSQ (formerly WSMQ) must specify PROGRAM(DFHWSDSQ).

BTS commands

The sample programs provided with this feature use a small subset of the large repertoire of BTS commands.

You can use the feature successfully without detailed knowledge of all BTS capabilities. If you base your programs on the sample programs provided, you can use most of the BTS commands in the samples with very few changes. Most of your changes will involve using BTS containers to pass data between the pipeline stages; the samples contain examples showing how to do this.

For more information about BTS, see the *CICS Business Transaction Services* manual.

The BTS commands used in the sample programs are:

CHECK ACQPROCESS
: Check the completion status of a process

CHECK ACTIVITY
: Check the completion status of an activity

DEFINE ACTIVITY
: Define a BTS activity

DEFINE INPUT EVENT
: Define a BTS input event

DEFINE PROCESS
: Define a BTS process

GET CONTAINER
: Retrieve data from a named data-container

LINK ACQPROCESS
: Execute a BTS process synchronously without context switching

LINK ACTIVITY
: Execute a BTS activity synchronously without context switching

MOVE CONTAINER
: Move data from one named data-container to another

PUT CONTAINER
: Save data in a named data-container

RETRIEVE REATTACH EVENT
: Retrieve the name of an event that caused the current activity to be reattached

RETURN ENDACTIVITY
: Return program control, ensuring that a BTS activity completes

Chapter 2. Pipelines and the SOAP for CICS feature

The SOAP for CICS feature consists of *pipelines*, which support *service providers* and *service requesters*.

Definitions

Pipeline
A sequence of programs arranged so that the output from one program is used as input to the next.

Service provider pipeline
A pipeline of user-provided and system-provided programs which receives an inbound SOAP message, processes the contents, and sends a response.

Service requester pipeline
A pipeline of user-provided and system-provided programs which sends an outbound SOAP message, receives the response, and processes the contents of the response.

What is a pipeline?

A pipeline is a sequence of processes arranged so that the output from one process is used as input to the next process. A pipeline is a useful model for processing messages that flow between a client and a server.

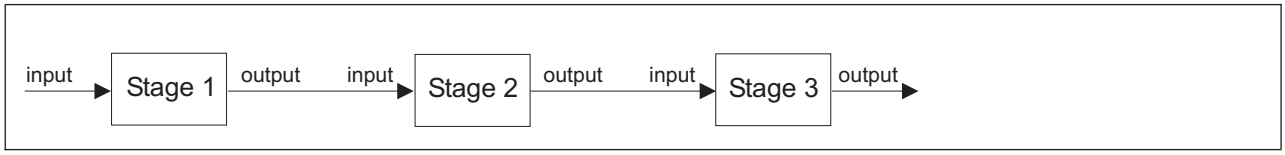


Figure 1. A pipeline as a series of processes

Figure 1 shows a simple pipeline with three processing stages; each processing stage contributes to the processing of an input to produce an output:

1. The input to the pipeline passes to stage 1 of the pipeline
2. The output from stage 1 passes to stage 2
3. The output from stage 2 passes to stage 3
4. The output from the final processing stage is the output from the pipeline as a whole

In a real system, a pipeline may be more complicated than this:

- If a processing stage detects an error, it may pass control to a special error handling process, rather than to the next processing stage.
- Each processing stage may use data which was not passed from the previous stage. For example, a processing stage might receive a message from a network.
- Each processing stage may create data which is not passed to the next stage. For example, a processing stage might store data in a file.
- Each processing stage may invoke programs which are not part of the pipeline, using a call-and-return mechanism. For example, a processing stage might call a

security manager to authenticate a user. Another processing stage might call an application program which is not part of the pipeline.

Using a pipeline to process messages

Although there are many situations in which a pipeline is a useful programming model (as well as many where it is not), we now focus on the particular example of processing messages that flow between a client and a server application. Typically, such messages are constructed of distinct layers; for example, one layer may contain information used by the target application, while others may contain information used by the server middleware.

A pipeline can effectively process this sort of message - for example, on input, each stage in the pipeline processes one layer, before removing it from the message, and passing the remaining layers to the next stage:

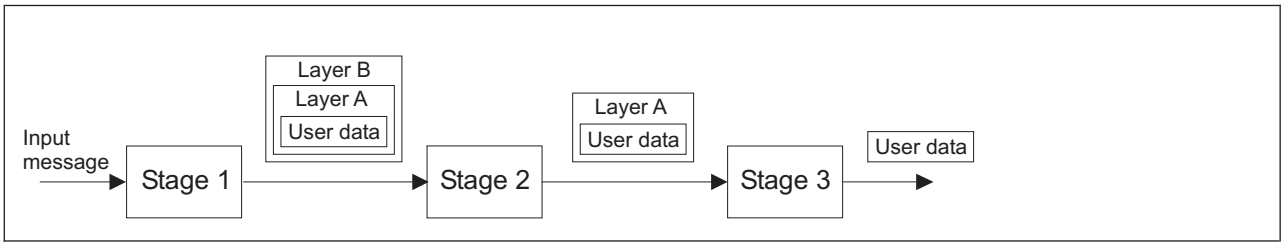


Figure 2. A pipeline used to process an input message

Figure 2 shows a three-stage pipeline applied to processing an input message from a network. The message is structured as user data wrapped in two layers of information used by the middleware (Layers A and B):

1. Stage 1 receives an input message from the network, and passes it to the next stage.
2. Stage 2 processes the outer layer of middleware information (Layer B), and passes the remaining layers to the next process.
3. Process 3 processes the next layer of middleware information (Layer A). The output from this stage, and from the pipeline as a whole, is the user data.

On output, the model operates in reverse: each stage in the pipeline constructs a layer and adds it to the message:

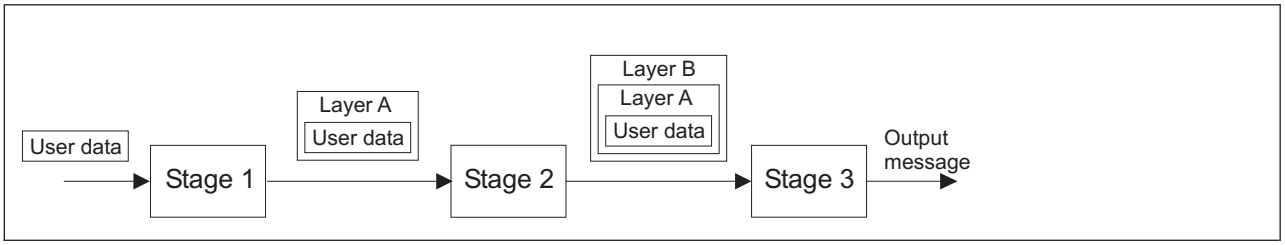


Figure 3. A pipeline used to process an output message

The pipeline model in a server

We have seen how separate pipelines can process an input message and an output message. By combining an input pipeline and an output pipeline, we can describe a server using the pipeline model:

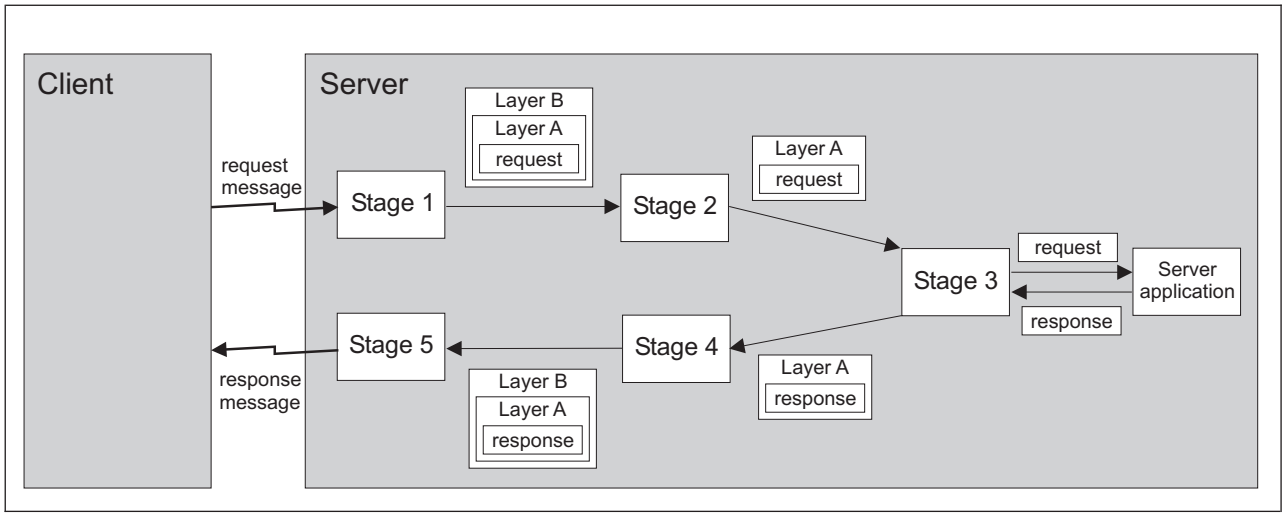


Figure 4. A server modeled as a pipeline

Figure 4 shows a five-stage pipeline which combines the input and output processing functions, and which calls an application program.

1. Stage 1 receives an input message from the client, and passes it to the next stage. The input message contains an application request, with two layers of middleware information.
2. Stage 2 processes the outer layer (Layer B), and passes the remaining layers to the next stage.
3. Stage 3 processes the next layer (Layer A), and calls the application program.
4. The application program processes the request, and returns a response.
5. Stage 3 adds the inner layer of middleware information (Layer A), and passes the message to the next stage.
6. Stage 4 adds the outer layer of middleware information (Layer B), and passes the message to the next stage.
7. Stage 5 send the output message to the client.

In this model, process 3 has a special place: it is the point at which processing of the input message ends, and processing of an output message starts. It is also the point at which the pipeline calls the target application. This stage in the pipeline is known as the *pivot point*.

The pipeline model in the client

We can also describe a client using the pipeline model, by concatenating an output pipeline and an input pipeline. In this model, we consider the work done by the server as a single processing stage.

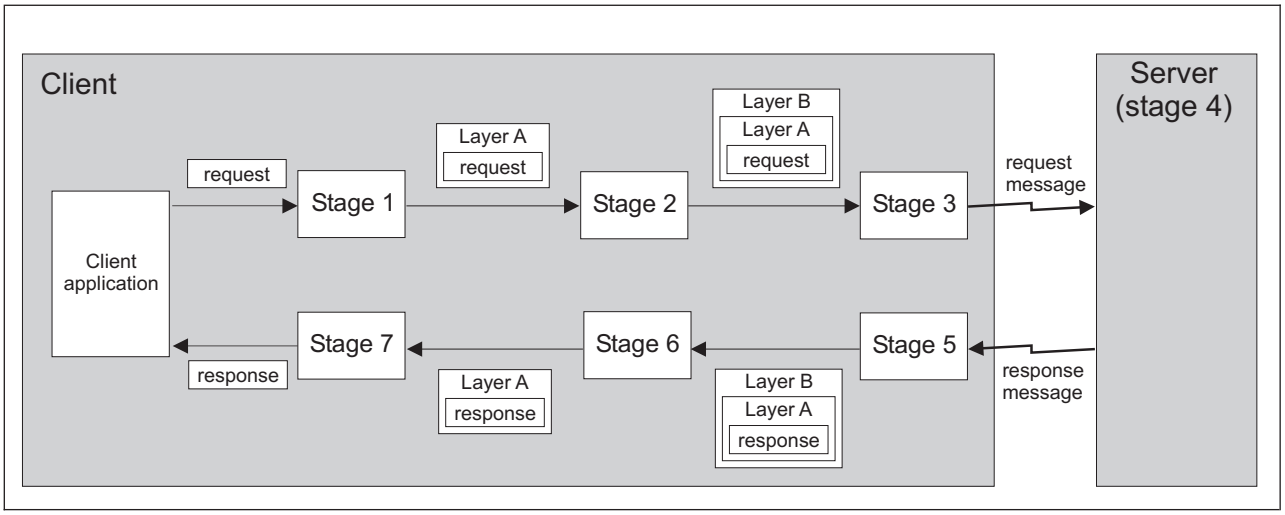


Figure 5. A client modeled as a pipeline

Figure 5 shows:

- a client application program
- three output stages
- the server, which - in this view - can be considered to be stage in the pipeline
- three input stages

1. The client application program passes the request to the first stage of the pipeline.
2. Stage 1 takes a request, adds the inner layer of middleware information (Layer A), and passes the message to the next stage.
3. Stage 2 adds the outer layer of middleware information (Layer B), and passes the message to the next stage.
4. Stage 3 send the message to the server.
5. The server (stage 4) is the pivot point of the pipeline; it processes the request message and returns a response message.
6. Stage 5 receives the response message from the server, and passes it to the next stage.
7. Stage 6 processes the outer layer (Layer B), and passes the remaining layers to the next stage.
8. Stage 7 processes the next layer (Layer A), and returns the response to the client application program.

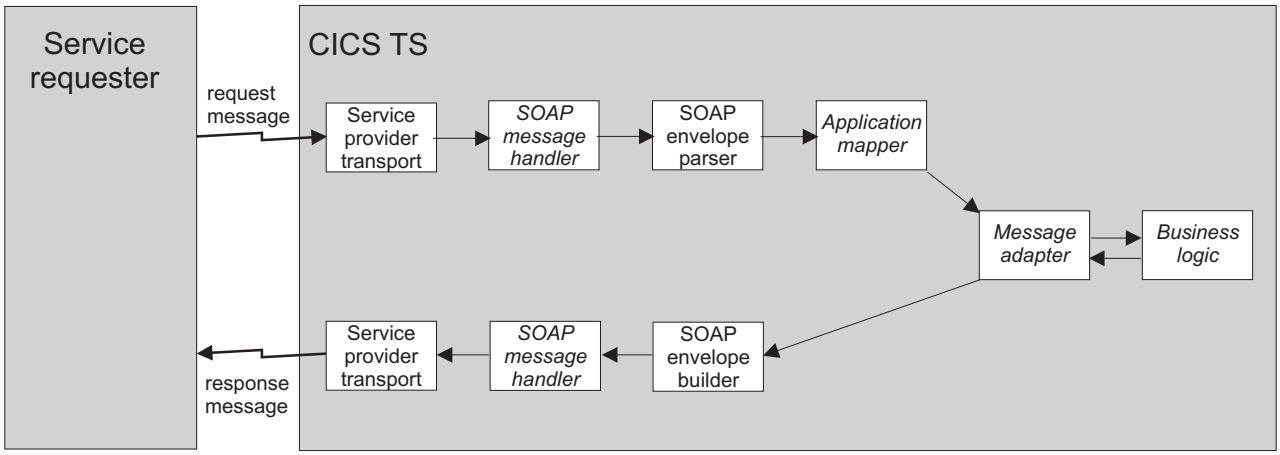
How SOAP for CICS uses pipelines

The SOAP for CICS feature is implemented using two pipelines.

The pipelines are

- The service provider pipeline
- The service requester pipeline

CICS as a service provider



Note: In this figure, italics represent user-written code.

Figure 6. The CICS SOAP service provider modeled as a pipeline

Figure 6 shows the pipeline stages which CICS uses to process a SOAP request, and invoke an application program:

- Some of the processing stages are provided with this feature, and you cannot change them
- Other processing stages are user-written code, which you may change to suit your needs (although the feature provides default actions for these stages).

The processing stages in this pipeline are:

The service provider transport stage (inbound)
This pipeline stage is responsible for extracting a SOAP request from an incoming message.

The SOAP for CICS feature provides a service provider transport program for the HTTP and WebSphere MQ message transports.

The SOAP message handler (user-written)
You can use this program to work with the complete SOAP input message. For example, you can extract information from the message, and modify its contents. For more information, see "Writing the SOAP message handler" on page 27.

The SOAP envelope parser
This pipeline stage extracts information from the SOAP request envelope, and removes the envelope from the incoming message.

The application mapper (user-written)
Use this program to specify the name of the message adapter program which is to be invoked for an inbound SOAP request. For more information, see "Writing the application mapper" on page 28.

The message adapter (user-written)
The message adapter is the interface between the pipeline and the business logic. The program:

1. Parses the SOAP request body
2. Invokes the business logic (typically, by using a LINK command with a commarea)
3. Constructs the SOAP response body

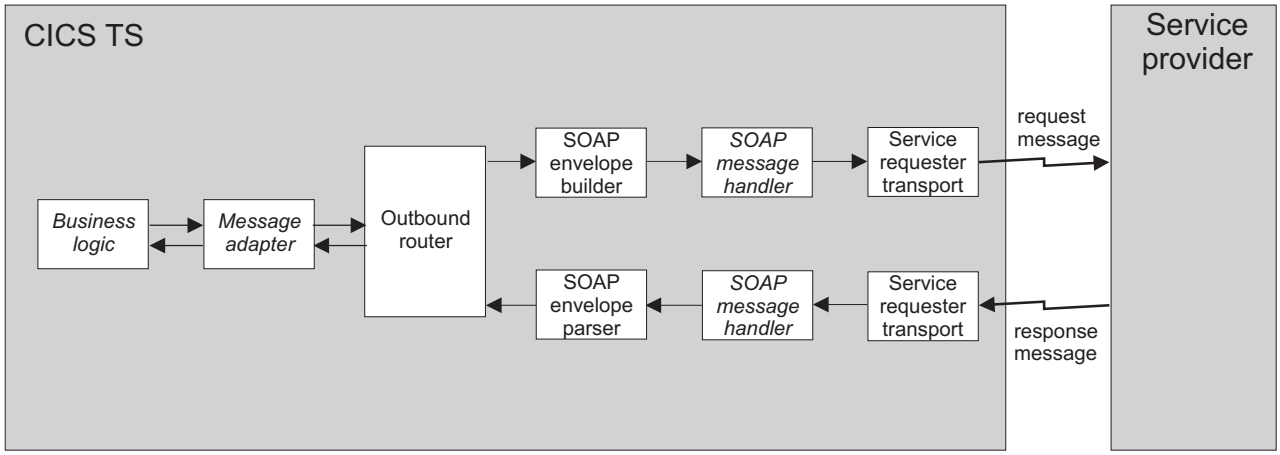
The SOAP envelope builder
This pipeline stage builds the SOAP response envelope, and adds it to the outgoing message.

The SOAP message handler (user-written)
You can use this program to work with the complete SOAP output message. For example, you can extract information from the message, and modify its contents. For more information, see "Writing the SOAP message handler" on page 27.

The service provider transport stage (outbound)
This pipeline stage is responsible for packaging a SOAP response within an outgoing message.

The SOAP for CICS feature provides a service provider transport program for the HTTP and WebSphere MQ message transports.

CICS as a service requester



Note: In this figure, italics represent user-written code.

Figure 7. The CICS SOAP service requester modeled as a pipeline

Figure 7 shows the pipeline stages which CICS create a SOAP request, and invoke a service provider:

- Some of the processing stages are provided with this feature, and you cannot change them
- Other processing stages are user-written code, which you may change to suit your needs (although the feature provides default actions for these stages).

The processing stages for a service requester are:

The business logic
Links to the message adapter

The message adapter (user-written)
The message adapter is the interface between the business logic and the pipeline. This program:

1. Constructs the SOAP request body
2. Invokes the outbound router program
3. Parses the SOAP response body

The outbound router program
The outbound router program passes the SOAP request body into the service requester pipeline, and retrieves the SOAP response from the pipeline.

The SOAP envelope builder
This pipeline stage builds the SOAP request envelope, and adds it to the outgoing message.

The SOAP message handler (user-written)
You can use this program to work with the complete SOAP output message. For example, you can extract information from the message, and modify its contents. For more information, see "Writing the SOAP message handler" on page 27.

The service requester transport stage (outbound)
This pipeline stage is responsible for packaging a SOAP request within an outgoing message.

The SOAP for CICS feature provides a service requester transport program for the HTTP and WebSphere MQ message transports.

The service provider
Receives and processes the SOAP request, and returns the response.

The service requester transport stage (inbound)
This pipeline stage is responsible for extracting a SOAP response from an incoming message.

The SOAP for CICS feature provides a service requester transport program for the HTTP and WebSphere MQ message transports.

The SOAP message handler (user-written)
You can use this program to work with the complete SOAP input message. For example, you can extract information from the message, and modify its contents. For more information, see "Writing the SOAP message handler" on page 27.

The SOAP envelope parser
This pipeline stage extracts information from the SOAP response envelope, and removes the envelope from the incoming message.

Chapter 3. Containers

The SOAP for CICS feature uses BTS containers to pass information between the processing stages in a pipeline.

You can use containers:

- to pass information from one user-written processing stage to another
- to pass information from a user-written stage and a CICS-supplied stage
- to pass information from a CICS-supplied stage and a user-written stage

Containers used by the feature have reserved names; you may not use these names for your own containers.

Important: The containers used in the service provider and service requester pipelines observe different naming conventions.

Containers used in the service provider pipeline

Container name	Purpose	More information
APP-HANDLER	Used to specify the name of the message adapter program	"Writing the application mapper" on page 28
INPUT	Used to pass the SOAP request or response body to the message handler and message adapter programs	"Writing a message adapter for the service provider pipeline" on page 19
NAMESPACES	Used to pass namespace information from the inbound SOAP envelope to the message adapter	"Writing a message adapter for the service provider pipeline" on page 19
OUTPUT	Use by the message handler and message adapter programs to return a modified SOAP request or response body to the pipeline	"Writing a message adapter for the service provider pipeline" on page 19
PIPELINE-ERROR	The presence of this container signals the existence of an error, and changes the sequence of processing of the pipeline. In user-written pipeline stages it can also be used to contain information about the error	"Error handling in the service provider pipeline" on page 31
TARGET-TRANID	Specifies the transaction ID under which the message adapter, and the application program should run	"Changing the application context" on page 21
TARGET-URI	Contains the URI of the service provider	
TARGET-USERID	Specifies the user ID under which the message adapter, and the application program should run	"Changing the application context" on page 21
USER-CONTAINERS	Used to pass data between user-written pipeline stages	"Passing data between user-written programs" on page 29

Containers used in the service requester pipeline

Container name	Purpose	More information
APP-NAMESPACES	Contains namespace information that is to be added to the outbound SOAP envelope	"Writing a message adapter for the service requester pipeline" on page 21
FAULT	Contains a SOAP fault which is constructed by the service requester pipeline	"Handling faults in the message adapter" on page 24
INPUT	Used to pass the SOAP request or response body to the message handler	"Writing a message adapter for the service requester pipeline" on page 21
OUTPUT	Use by the message handler program to return a modified SOAP request or response body to the pipeline	"Writing a message adapter for the service requester pipeline" on page 21
PIPELINE-ERROR	The presence of this container signals the existence of an error, and changes the sequence of processing of the pipeline. In user-written pipeline stages it can also be used to contain information about the error	"Error handling in the service requester pipeline" on page 33
REQUEST-BODY	Contains the body of the SOAP request that is to be passed to the service provider	"Writing a message adapter for the service requester pipeline" on page 21
RESPONSE-BODY	Contains the SOAP response body that is returned by the service provider	"Writing a message adapter for the service requester pipeline" on page 21
SOAP-ACTION	For the HTTP transport only, specifies the contents of the HTTP SOAPAction header	"Writing a message adapter for the service requester pipeline" on page 21
TARGET-URI	Specifies the URI of the target service provider	"Writing a message adapter for the service requester pipeline" on page 21
USER-CONTAINERS	Used to pass data between user-written pipeline stages	"Passing data between user-written programs" on page 29

Containers used by user-written programs

This topic explains which containers are available to each of the user-written programs in the service provider and service requester pipelines.

Notation

The following notation is used in this topic:

Expected input
The container exists on entry to the user-written program.

Possible input
The container may or may not exist on entry to the user-written program. The program should test whether the container exists.

Required output
The container must exist on exit from the user-written program. If the container does not exist on entry, the program must create it.

Optional output
The container need not exist on exit from the user-written program. If the container exists on entry, the program may delete it.

Not available
The container is not available to the user-written program.

Containers in the service provider pipeline

The following table shows which containers are available to each user-written program in the service provider pipeline.

Container	User-written program			
	Message handler (inbound)	**Application mapper**	**Message adapter**	**Message handler (outbound)**
APP-HANDLER	Not available	Expected input Optional output	Not available	Not available
INPUT	Expected input	Not available	Expected input	Expected input
NAMESPACES	Not available	Not available	Possible input Optional output	Not available
OUTPUT	Optional output	Not available	Optional output	Optional output
PIPELINE-ERROR	Optional output	Optional output	Optional output	Possible input
TARGET-TRANID	Optional output	Possible input Optional output	Expected input	Expected input
TARGET-URI	Expected input	Expected input	Expected input	Not available
TARGET-USERID	Optional output	Possible input Optional output	Expected input	Expected input
USER-CONTAINERS	Optional output	Possible input Optional output	Possible input Optional output	Possible input

Note: On input, the pipeline uses two different containers to carry the SOAP request:

- The message handler receives the SOAP request in container INPUT; if the handler modifies the request, it returns it to the pipeline in container OUTPUT.
- The message adapter receives the request body in container INPUT.

Likewise, on output two different containers are used to carry the SOAP response:

- When the message adapter constructs the response body, it passes it to the pipeline in container OUTPUT.
- The message handler receives the SOAP response in container INPUT; if the handler modifies the response, it returns it to the pipeline in container OUTPUT.

Containers in the service requester pipeline

The following table shows which containers are available to each user-written program in the service requester pipeline.

Container	User-written program			
	Message adapter (outbound)	**Message handler (outbound)**	**Message handler (inbound)**	**Message adapter (inbound)**
APP-NAMESPACES	Optional output	Optional output	Possible input	Possible input
FAULT	Not available	Not available	Possible input	Possible input
INPUT	Not available	Expected input	Expected input	Expected input

Container	User-written program			
	Message adapter (outbound)	**Message handler (outbound)**	**Message handler (inbound)**	**Message adapter (inbound)**
OUTPUT	Not available	Optional output	Optional output	Not available
PIPELINE-ERROR	Optional output	Possible input Optional output	Possible input Optional output	Possible input
REQUEST-BODY	Required output	Not available	Not available	Not available
RESPONSE-BODY	Not available	Not available	Not available	Expected input
SOAP-ACTION	Optional output	Optional output	Possible input	Possible input
TARGET-URI	Required output	Optional output	Not available	Possible input
USER-CONTAINERS	Optional output	Possible input Optional output	Possible input Optional output	Possible input

Note: On output, the pipeline uses three different containers to carry the SOAP request:

- When the message adapter constructs the request body, it passes it to the pipeline in container REQUEST-BODY.
- The message handler receives the SOAP request in container INPUT; if the handler modifies the request, it returns it to the pipeline in container OUTPUT.

Likewise, on input three different containers are used to carry the SOAP response:

- The message handler receives the SOAP response in container INPUT; if the handler modifies the response, it returns it to the pipeline in container OUTPUT.
- The message adapter receives the response body in container RESPONSE-BODY.

Chapter 4. Writing message adapters

A message adapter is the interface between an application's business logic, and the SOAP pipeline.

- In a service provider, the message adapter calls the business logic
- In a service requester, the business logic calls the message adapter

It is advisable to have the message adapter and business logic in different programs, and have one program LINK to the other.

Message adapters use CICS business transaction services (BTS) as the interface with the SOAP support provided by this feature. For example:

- in a service provider, the inbound SOAP request body is passed to the adapter in a container, and the adapter returns the outbound SOAP response body in another container.
- in a service requester, the adapter constructs the outbound SOAP request body in a container, and the response is returned to the adapter in another container.

You should be familiar with the differences between BTS applications and other CICS applications before you write a SOAP message adapter. See the *CICS Business Transaction Services* manual for more information.

Restriction: In a service provider pipeline that uses the WebSphere MQ transport, neither the message adapter, nor the business logic which the adapter invokes, can issue the EXEC CICS SYNCPOINT command.

Writing a message adapter for the service provider pipeline

These are the main processing steps that your message adapter should perform.

1. Retrieve the attach event, with the RETRIEVE REATTACH EVENT command. The adapter program is a BTS activity program, which is attached one time by the pipeline manager. All BTS activity programs must deal with their reattachment events.
2. Optional: If your adapter expects the inbound SOAP body to contain namespace-qualified element and attribute names, use the GET CONTAINER command to retrieve the namespace information from the NAMESPACES container. For example:

   ```
   EXEC CICS GET CONTAINER('NAMESPACES')
                 SET(NSP-PTR)
                 FLENGTH(NSP-LEN)
   ```

   The namespace information consists of a list of space-separated namespace definitions. Each definition has the form *symbol*="*URI*". For example

   ```
   SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" svg="http://www.w3.org/2000/svg"
   ```

   Note: In a SOAP message, the namespace prefix may be declared, and bound to a namespace URI, in the SOAP envelope, or in the SOAP body. Only namespace information that is declared in the SOAP envelope is passed to the adapter program in the NAMESPACES container. Your adapter is responsible for detecting namespace information that is declared within the SOAP body.

3. Use the GET CONTAINER command to retrieve the request body from the INPUT container. For example:

```
EXEC CICS GET CONTAINER('INPUT')
              SET(BODY-PTR)
              FLENGTH(BODY-LEN)
```

   The request body includes the <SOAP-ENV:body> element. For example:

```
<SOAP-ENV:body>
  <symbol>
    XYZ
  </symbol>
</SOAP-ENV:body>
```

4. Parse the request body. You can use the language statements which some compilers provide for parsing XML, you can LINK to a parsing program, or you can provide your own parsing code in the body of your program.
5. Using the relevant data from the request body, invoke the business logic. It is advisable to keep the business logic and the manipulation of the SOAP message separated. To do this, put the business logic in a separate program, and LINK to it.
6. Using the response from the business logic, construct the body of the SOAP response. For example:

```
<SOAP-ENV:body>
  <Quote>130</Quote>
  <Date>12/31/03</Date>
  <Time>10:43:21</Time>
</SOAP-ENV:body>
```

   Tip: You can use the CICS DOCUMENT API to construct the SOAP body.
7. Optional: If your adapter constructs a SOAP body that contains namespace-qualified element and attribute names, and you want the namespace prefix to be declared, and bound to a namespace URI, in the SOAP envelope, pass the information in the NAMESPACES container, using the PUT CONTAINER command. For example:

```
EXEC CICS PUT CONTAINER('NAMESPACES')
              FROM(OUT-NSP)
              FLENGTH(OUT-NSP-LEN)
```

   The namespace information consists of a list of space-separated namespace definitions. Each definition has the form *symbol*="*URI*". For example:

```
SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" svg="http://www.w3.org/2000/svg"
```

   Notes:

   - In a SOAP message, the namespace prefix may be declared, and bound to a namespace URI, in the SOAP envelope, or in the SOAP body. If you do not declare the namespace information in the SOAP envelope, your adapter is responsible for providing the namespace information within the SOAP body.
   - Your adapter can assume that the prefix SOAP-ENV will be bound to http://schemas.xmlsoap.org/soap/envelope in the outbound SOAP envelope; you do not need to pass this declaration in the container.
8. Return the response body to the pipeline in the OUTPUT container, using the PUT CONTAINER command. For example:

```
EXEC CICS PUT CONTAINER('OUTPUT')
              FROM(OUT-BODY)
              FLENGTH(OUT-BODY-LEN)
```

Changing the application context

Optionally, in a service provider pipeline, you can specify that the message adapter, and the application program that it calls, should run in a different context to the rest of the pipeline:

- They can run under a different transaction ID
- They can run under a different user ID.
- They can run under a different transaction ID **and** user ID.

In all three cases, the message adapter and the application program run under a different task.

Changing the transaction ID

To specify a different transaction ID, code the following steps in your application mapper program (recommended) or in your inbound SOAP message handler.

1. Create a container called TARGET-TRANID.
2. Put the transaction ID of the target transaction in the TARGET-TRANID container. The transaction ID should be in the first four characters of the data. If you supply more than four characters, the data is truncated; if you supply fewer than four, the data is padded with blanks.

   Restriction: The target transaction must be a local transaction: you cannot specify the name of a remote or dynamically-routed transaction.

The message adapter and the application program will run in a new task, with the transaction ID specified.

Changing the user ID

Before you can change the user ID, you must specify in your security manager that the original user ID under which the pipeline runs is a surrogate of the target user ID (the user ID under which the transaction the message adapter and application program should run). For more information, see the *CICS RACF Security Guide*.

To specify a different user ID, code the following steps in your application mapper program (recommended) or in your inbound SOAP message handler.

1. Create a container called TARGET-USERID.
2. Put the user ID under which the message adapter and application program should run in the TARGET-USERID container. The user ID should be in the first eight characters of the data. If you supply more than eight characters, the data is truncated; if you supply fewer than eight, the data is padded with blanks.

The message adapter and the application program will run in a new task, under the user ID specified.

Writing a message adapter for the service requester pipeline

These are the main processing steps that your message adapter should perform.

1. Construct the outbound SOAP request body
2. Invoke the outbound SOAP router program
3. Process the inbound SOAP response

Constructing the SOAP request

Depending upon the requirements of your adapter, you may need to construct a root activity, and child activities. The steps described assume that you will perform all parts of your adapter in the root activity.

1. Use the DEFINE PROCESS command to add a new CICS business transaction services (BTS) process, and create the root activity.
   - The process name must be unique within the scope of the process type.
   - Specify the name of the outbound SOAP router program (DFHWSRT) in the PROGRAM option of the command.
   - For the best performance when using the HTTP transport, specify the NOCHECK option. You can also specify this option with the WebSphere MQ transport, but it has no effect on performance.

   For example:

   ```
   EXEC CICS DEFINE PROCESS(PROCESS-NAME)
                    PROCESSTYPE('SOAPHTTP')
                    TRANSID('CSAC')
                    PROGRAM('DFHWSRT')
                    NOCHECK
   ```

2. Pass the URI of the target service provider code to the outbound SOAP router program in container TARGET-URI, using the PUT CONTAINER command. For example:

   ```
   EXEC CICS PUT CONTAINER('TARGET-URI')
                 ACQACTIVITY
                 FROM(TARGET)
                 FLENGTH(TARGET-LEN)
   ```

   - For the HTTP transport, the target is specified as the URI of the target process. For example:

     `http://ip-address/CICS/CWBA/DFHWSDSH/DFH$WSSS`

   - For the WebSphere MQ transport, the target is specified in the following form:

     `MQ://queuename/queue_manager`

     where the queue manager name is optional if the queue is local, or a remote queue definition exists. For example:

     `MQ://SAMPLE.SOAP.QUEUE/QM39`

3. Using information obtained from the business logic, construct the body of the SOAP request. The message body must include the <SOAP-ENV:body> element. For example:

   ```
   <SOAP-ENV:body>
     <symbol>
       XYZ
     </symbol
   </SOAP-ENV:body>
   ```

4. Pass the SOAP request body to the outbound SOAP router program in container REQUEST-BODY, using the PUT CONTAINER command. For example:

   ```
   EXEC CICS PUT CONTAINER('REQUEST-BODY')
                 ACQACTIVITY
                 FROM(OUT-BODY)
                 FLENGTH(OUT-BODY-LEN)
   ```

5. Optional: If your adapter constructs a SOAP request body that contains namespace-qualified element and attribute names, and you want the namespace prefix to be declared, and bound to a namespace URI, in the SOAP

envelope, pass the namespace information to the outbound SOAP router program in container APP-NAMESPACES, using the PUT CONTAINER command. For example:

```
EXEC CICS PUT CONTAINER('APP-NAMESPACES')
             ACQACTIVITY
             FROM(OUT-NSP)
             FLENGTH(OUT-NSP-LEN)
```

The namespace information consists of a list of space-separated namespace definitions. Each definition has the form *symbol*="*URI*". For example

```
SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" svg="http://www.w3.org/2000/svg"
```

Notes:

- In a SOAP message, the namespace prefix may be declared, and bound to a namespace URI, in the SOAP envelope, or in the SOAP body. If you do not declare the namespace information in the SOAP envelope, your adapter is responsible for providing the namespace information within the SOAP body.
- Your adapter can assume that the prefix `SOAP-ENV` will be bound to `http://schemas.xmlsoap.org/soap/envelope` in the outbound SOAP envelope; you do not need to pass this declaration in the container.

6. Optional: For the HTTP transport, pass the SOAPAction header to the outbound SOAP router program in container SOAP-ACTION, using the PUT CONTAINER command. For example:

```
EXEC CICS PUT CONTAINER('SOAP-ACTION')
             ACQACTIVITY
             FROM(ACTION-HEADER)
             FLENGTH(ACTION_BODY-LEN)
```

You can specify a SOAPAction header of up to 256 characters in length. If you do not specify a SOAPAction header, the feature creates an empty header.

Invoking the outbound SOAP router program

1. Execute the SOAP router process that was acquired in “Constructing the SOAP request” on page 22. Use the following command to execute the process synchronously:

```
EXEC CICS LINK ACQPROCESS
```

2. Use the CHECK ACQPROCESS command to determine the completion status of the process. For example:

```
EXEC CICS CHECK ACQPROCESS
               COMPSTATUS(PROCESS-STATUS)
               ABCODE(PROCESS-ABCODE)
```

Processing the SOAP response

If the SOAP request was processed successfully, there will be a SOAP response body to process. However, if the request was not processed successfully, there may be a SOAP fault to process instead.

1. Process any fault information returned in the FAULT container. For more information, see “Writing a message adapter for the service requester pipeline” on page 21

2. If your adapter expects the inbound SOAP response body to contain namespace-qualified element and attribute names, retrieve the namespace information from the APP-NAMESPACES container, with the GET CONTAINER command. For example:

```
EXEC CICS GET CONTAINER('APP-NAMESPACES')
              SET(NSP-PTR)
              FLENGTH(NSP-LEN)
```

   Notes:

   - In a SOAP message, the namespace prefix may be declared, and bound to a namespace URI, in the SOAP envelope, or in the SOAP body. Only namespace information that is declared in the SOAP envelope is passed to the adapter program in the container. Your adapter is responsible for detecting namespace information that is declared within the SOAP body.
   - Your adapter can assume that the prefix `SOAP-ENV` is bound to `http://schemas.xmlsoap.org/soap/envelope`. If this is the only namespace of interest to your adapter, you do not need to retrieve the namespace information from the container.

3. Retrieve the response body from the RESPONSE-BODY container, with the GET CONTAINER command. For example:

```
EXEC CICS GET CONTAINER('RESPONSE-BODY')
             SET(BODY-PTR)
             FLENGTH(BODY-LEN)
```

   The response body includes the `<SOAP-ENV:body>` element. For example:

```
<SOAP-ENV:body>
  <Quote>130</Quote>
  <Date>12/31/03</Date>
  <Time>10:43:21</Time>
</SOAP-ENV:body>
```

4. Parse the SOAP response body. You can use the language statements which some compilers provide for parsing XML, you can link to a parsing program, or you can provide your own parsing code in the body of your program.
5. Using the relevant data from the response body, invoke the business logic to process the response. It is advisable to keep the business logic and the manipulation of the SOAP message separated. To do this, put the business logic in a separate program, and LINK to it.

Handling faults in the message adapter

In the service requester pipeline, SOAP faults can be returned to the message adapter in two situations:

The service provider returns an invalid SOAP response
: In this case, the service requester pipeline detects the error, and returns the fault to the adapter in the FAULT container. Because the SOAP response is invalid, the RESPONSE-BODY and APP-NAMESPACES containers will be empty.

The service provider returns a fault
: In this case, the SOAP fault element is returned to the service requester in the response body. Your message adapter program must detect the fault element, and process it if necessary.

In general, you should write your message adapter to deal with both situations.

1. Before attempting to process the response body, retrieve any fault information from the FAULT container, with the GET CONTAINER command. For example:

```
EXEC CICS GET CONTAINER('FAULT')
              SET(FAULT-PTR)
              FLENGTH(FAULT-LEN)
```

   - If the container does not exist (that is, the command returns the CONTAINERERR condition), the SOAP response is valid, and your program can continue by processing the response body.
   - If the container is empty (that is, the returned length is zero), the SOAP response is valid, and your program can continue by processing the response body.
   - If the container is not empty, there is fault information in the container, in a `<SOAP-ENV:Fault>` element. In this case, there is no response body to process, and the length of the RESPONSE-BODY container is zero.
2. When your program parses the response body response, it should detect the `<SOAP-ENV:Fault>` element.

If your program detects a fault, of either sort, it can parse the `<SOAP-ENV:Fault>` element to extract further information about the error, and take appropriate action.

Identifying the target of a SOAP message

When an application sends a SOAP request, it must specify where the request will be processed, and the name of the program that will process it. The way the target is identified depends upon the transport protocol used.

Identifying the target for the HTTP transport

The target is identified using an HTTP Universal Resource Identifier (URI):

- If you are sending a SOAP message to a non-CICS SOAP server, use the URI of the target in that server.
- If you are sending a SOAP message to a service provider application in a CICS region, the URI is interpreted by the analyzer program used by CICS Web support. Details of how to write an analyzer program are given in the *CICS Internet Guide*.

  If you use the default analyzer, the URI has the following form:

  `http://host:port/CICS/CWBA/DFHWSDSH/program`

  where *program* is the name of the target program.

Identifying the target for the WebSphere MQ transport

The target is identified using a string in the following form:

`MQ://queue_name/queue_manager`

where *queue_name* is the name of a queue that is associated with a trigger process. *queue_manager* is optional if the queue is local, or if a remote queue definition exists.

The target program name is specified in the trigger data of the target queue.

Chapter 5. Writing your own pipeline programs

The pipelines that process SOAP messages include programs in which you can provide your own processing. They are:

The SOAP message handler (DFHWSHDX)
Use this program to work with the complete SOAP input or output message. For example, you could use this program to process SOAP headers that the SOAP for CICS feature code does not process.

The application mapper (DFHWSAMX)
Use this program to specify the name of the message adapter program.

In any CICS region, there is just one SOAP message handler, and just one application mapper. Both programs are always invoked at the appropriate points in the pipeline, for every request. Therefore you must design your programs accordingly:

- Avoid unnecessary processing in the programs where this may affect all your SOAP applications. Design your programs to relinquish control as quickly as possible in those cases where no processing is required.
- Where possible, avoid application-specific processing of the SOAP message in the programs. For service providers, it is advisable to use the message adapter for this purpose.
- Where application-specific code is unavoidable, structure your program to accommodate code for all applications that may use the program. Design the program to be easily extended.

The feature includes sample of these programs which do nothing other than return control to the pipeline immediately. Use these samples if you do not need to write your own code. The samples are provided as source code and as object modules.

Writing the SOAP message handler

Use the SOAP message handler (DFHWSHDX) to work with the complete SOAP input or output message. For example, you can use this handler to extract information from the message, and to modify its contents.

1. Determine at which stage the program is being called. The SOAP message handler is invoked at the following points:

   For a service provider application
   - When the SOAP request is received
   - Before the SOAP response is sent

   For a service requester application
   - Before the SOAP request is sent
   - When the SOAP response is received

   To distinguish between these cases, check the BTS activity name, and the input event:

Activity name	Input event	Service provider or requester?	Type of message
USERHANDLER-IN	DFHINITIAL	Service provider	SOAP request (input)
USERHANDLER-IN	SEND-RESPONSE	Service provider	SOAP response (output)
USERHANDLER-OUT	DFHINITIAL	Service requester	SOAP request (output)
USERHANDLER-OUT	RECEIVE-RESPONSE	Service requester	SOAP response (input)

- To determine the activity name, use the ACTIVITY option of the ASSIGN command. For example:

  ```
  EXEC CICS ASSIGN ACTIVITY(ACTIVITY-NAME)
  ```

- To determine the input event, use the RETRIEVE REATTACH EVENT command. For example:

  ```
  EXEC CICS RETRIEVE REATTACH EVENT(EVENT-NAME)
  ```

2. The first time the message handler is invoked in a pipeline, define the input event which will activate the message handler for the second time. To define the input event, use the DEFINE INPUT EVENT command.
   - For a service provider application, the input event is SEND-RESPONSE
   - For a service requester application, the input event is RECEIVE-RESPONSE

   For example:

   ```
   EXEC CICS DEFINE INPUT EVENT(EVENT-NAME)
   ```

   Do not issue this command the second time the message handler is invoked.
3. Include any code that you need to process the SOAP request or response. The SOAP message is in container INPUT.
4. Determine what further processing is required for this request or response.
5. Optional: If you have modified the SOAP message, return it to the pipeline in container OUTPUT. If you do not return this container, the pipeline processes the original message.
6. Use container USER-CONTAINERS to pass other data to the next stage in the pipeline.
7. Return control to CICS
   - The first time the program is invoked in a pipeline, use EXEC CICS RETURN.
   - The second time the program is invoked, use EXEC CICS RETURN ENDACTIVITY.

Writing the application mapper

Use the application mapper (DFHWSAMX) to specify the name of the message adapter that is to be invoked by the service provider pipeline.

1. Include any code you need to determine the name of the target message adapter program. The original name of the target is in container APP-HANDLER.

   To determine the name of the target, you may need to use information which is determined at an earlier stage of the pipeline. For more information, see “Passing data between user-written programs” on page 29. You can also fine the URI of the original request in container TARGET-URI.

2. Optional: To change the name of your application program, return the new name to the pipeline in container APP-HANDLER. If you do not return this container, the pipeline uses the original name.

Passing data between user-written programs

In some situations, you will want to pass your own data from one user-written program (the *data source*) to another (the *data target*). To do this, use one or more BTS containers. The feature provides another container, called USER-CONTAINERS in which you specify the names of your own containers:

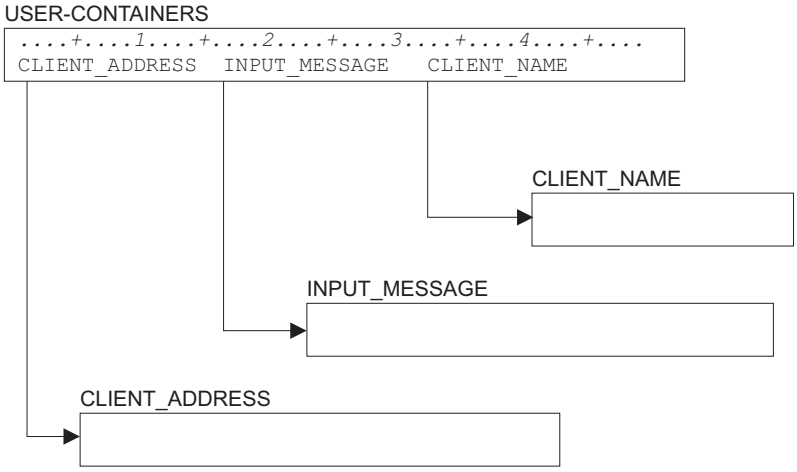


Figure 8. Using USER-CONTAINERS to pass the names of your own containers. This figure illustrates the use of the USER-CONTAINERS container to pass the names of three other containers (CLIENT_ADDRESS, INPUT_MESSAGE, and CLIENT_NAME) between user-written programs.

Important: Do not attempt to use a different way of sharing container names between programs (such as hard-coding the container names in the application program). The data source and target programs may run under different BTS activities, and the feature uses the contents of USER-CONTAINERS to create containers in the target activity that have the same names (and contents) as those in the source.

1. In the data source, use the PUT CONTAINER command to store your data in one or more containers. For example:

```
EXEC CICS PUT CONTAINER('CLIENT_ADDRESS')
              FROM(CLIENT-ADDR)
              FLENGTH(CLIENT-ADDR-LEN)
```

   You can name the containers to suit your application , but do not use the names which are used by the feature for its containers.

2. Construct an array of the container names you used in the previous step.
   - Each element in the array must be exactly 16 bytes in length
   - Each container name in the array must be padded on the right with blanks to a length of 16 bytes
   - Elements in the array that do not contain container names must be filled with blanks

   For example:

```
....+....1....+....2....+....3....+....4....+....
CLIENT_ADDRESS  INPUT_MESSAGE   CLIENT_NAME
```

3. Pass the list of container names to the pipeline in container USER-CONTAINERS, using the PUT CONTAINER command. For example:

```
EXEC CICS PUT CONTAINER('USER-CONTAINERS')
              FROM(CONTAINER-LIST)
              FLENGTH(CONT-LIST-LEN)
```

4. In the data target, use the GET CONTAINER command to retrieve the list of containers from the container USER-CONTAINERS. For example:

```
EXEC CICS GET CONTAINER('USER-CONTAINERS')
              INTO(CONTAINER-LIST)
```

   Important: The feature may change the position and sequence of items in the list as passes from one program to the next. In particular, the names of non-existent containers may be removed. Therefore, your code should not rely on the position of an item in the list in any way.

5. Use the GET CONTAINER command to retrieve your data from a named container. For example:

```
EXEC CICS GET CONTAINER('CLIENT_ADDRESS')
              INTO(CLIENT-ADDR)
```

Chapter 6. Error handling in SOAP for CICS feature

When the SOAP for CICS feature detects an error in a processing stage of a pipeline, and it is not possible to complete normal processing, the sequence of execution of the pipeline changes.

The SOAP for CICS feature handles errors in a processing stage by abending the task. Subsequent processing proceeds in one of two ways:

- For most errors, some of the pipeline processing stages are omitted, and processing continues at a stage where recovery action can be sensibly performed. Depending upon where in the pipeline the error was detected, subsequent processing may include the execution of user-written programs:
  - If a user-written program is invoked, it may be able to perform some recovery processing of its own.
  - If a user-written program is not invoked, the feature performs a default recovery action.
- For some errors (specifically those abends which cannot be handled in the normal way), pipeline processing terminates immediately.

Handling errors in user-written programs

You can handle errors in your user-written programs in the following ways.

1. Make sure that your program contains HANDLE ABEND commands that will deal with any errors that may occur in your code. If you do not do so, and an abend occurs, the entire pipeline will abend.
2. If your program cannot recover from the error and allow the pipeline to continue normally, create a container called PIPELINE-ERROR. When the pipeline detects the presence of this container, subsequent processing follows the error path. The contents of the container are not used by the pipeline, but you can use it to pass information to any other user-written programs on the error path.

Error handling in the service provider pipeline

If an error is detected in the service provider pipeline, the sequence of processing follows the appropriate error path shown in Figure 9 on page 32. For example, if the SOAP envelope parser detects an error, control passes directly to the SOAP envelope builder, bypassing the application mapper, the message adapter, and the application program. The outbound SOAP message handler, and the outbound transport program are invoked in the normal way.

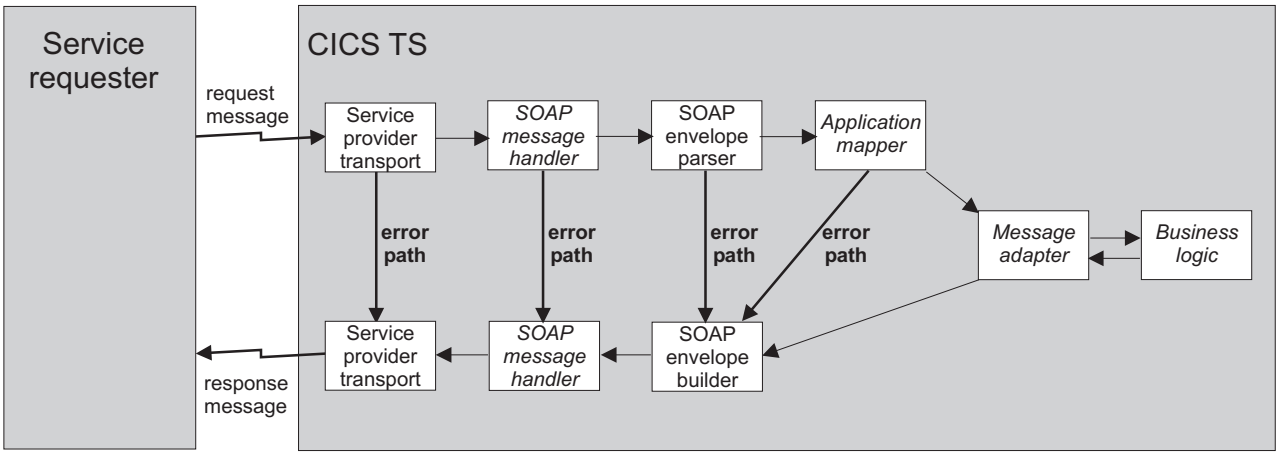


Figure 9. Error handling in a service provider

Pipeline stage in which error occurred	Pipeline stage to which control passes	Recovery action
Inbound service provider transport	Outbound service provider transport	Default error recovery
Inbound SOAP message handler	Outbound SOAP message handler	Determined by the user-written SOAP message handler
SOAP envelope parser	SOAP envelope builder	Default error recovery. However, the outbound SOAP message handler can modify the SOAP response generated by the default recovery action
Application mapper	SOAP envelope builder	Normal processing continues, but there is no SOAP response body

Default error recovery in the service provider pipeline

In the absence of other error recovery by the feature or by user-written programs, the outbound transport program constructs a SOAP response which contains a SOAP `<fault>` element, and sends it to the service requester. The `<fault>` element contains the following `<detail>` element:

```
<detail>
 <abend xmlns="http://cts.software.ibm.com/cicsts/soap/">
  <abcode>abcode</abcode>
  <program>abprog</program>
 </abend>
</detail>
```

where:

abcode
: is the abend code which caused the fault

abprog
: is the name of the CICS program which abended

When the transport is HTTP, the SOAP response is accompanied by an HTTP status code of `500 Internal Server Error`.

For the WebSphere MQ transport only, the pipeline transaction abends with code AWSB on completion of the default error handling.

Performing error recovery in the SOAP message handler

Depending upon where in the service provider pipeline an error occurred, you can use the outbound SOAP message handler to detect and handle the error.

1. Determine whether an error has occurred in the pipeline. When an error occurs, the feature creates a container called PIPELINE-ERROR. To detect the error, use the INQUIRE CONTAINER command. For example:

   ```
   EXEC CICS INQUIRE CONTAINER('PIPELINE-ERROR')
   ```

   If the command completes successfully, the container exists, indicating that a previous pipeline stage has encountered an error. If the command returns a CONTAINERERR condition, the container does not exist, indicating that all previous pipeline stages have completed normally.
2. Construct a suitable SOAP response body. Depending upon where in the pipeline the error occurred, container INPUT may contain the default SOAP response body containing a `<fault>` element.
3. Optional: Delete container PIPELINE-ERROR.
   - If you delete the container, the outbound transport program completes pipeline processing normally, as if no error had occurred. In this case, the output from the message handler must be a valid SOAP response.
   - If you do not delete the container, the pipeline ends abnormally. In this case, the output from the message handler must be a valid SOAP response containing a `<fault>` element.

   Depending upon where the error occurred, container INPUT may contain a suitable SOAP response. If it does not, or you want to change it, return your response in container OUTPUT.

Error handling in the service requester pipeline

If an error is detected in the service requester pipeline, the sequence of processing follows the appropriate error path shown in Figure 10 on page 34. For example, if the outbound SOAP message handler detects an error, control passes directly to inbound SOAP message handler, and no attempt is made to send a request to the service provider. The SOAP envelope parser is invoked in the normal way.

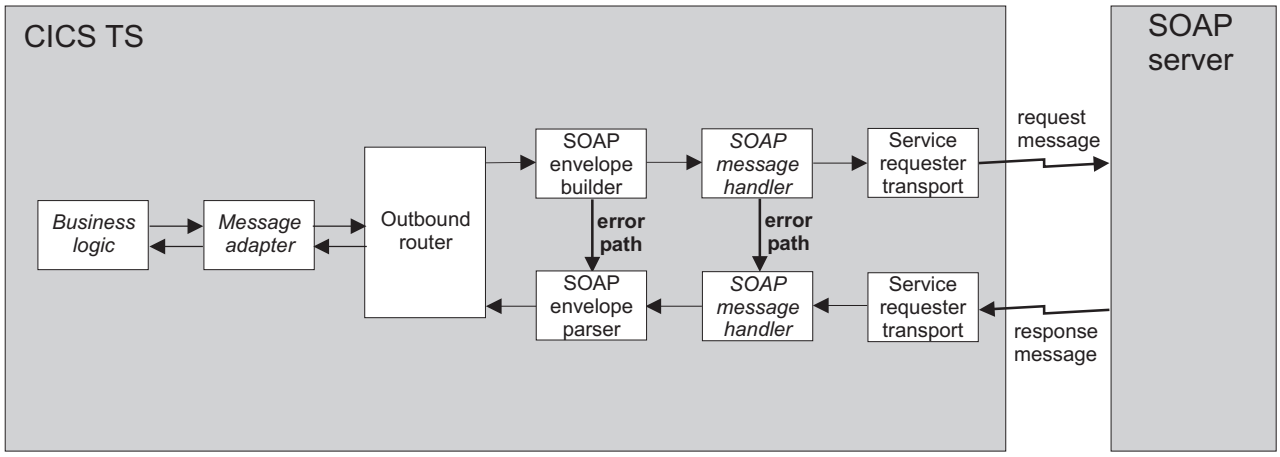


Figure 10. Error handling for a service requester

Pipeline stage in which error occurred	Pipeline stage to which control passes	Recovery action
Outbound SOAP message handler	Inbound SOAP message handler	Determined by the user-written SOAP message handler

Performing error recovery in the SOAP message handler

When an error occurs in the service requester pipeline, you can use the inbound SOAP message handler to detect and handle the error.

1. Determine whether an error has occurred in the pipeline. When an error occurs, the outbound SOAP message handler creates a container called PIPELINE-ERROR. To detect the error, use the INQUIRE CONTAINER command. For example:

   ```
   EXEC CICS INQUIRE CONTAINER('PIPELINE-ERROR')
   ```

   If the command completes successfully, the container exists, indicating that the outbound SOAP message handler has encountered an error. If the command returns a CONTAINERERR condition, the container does not exist, indicating that all previous pipeline stages have completed normally.
2. Construct a suitable SOAP response body. Pass the response body to the next pipeline stage in container RESPONSE-BODY.

Abend codes

The SOAP for CICS feature may generate the following abend codes.

AWSC

Explanation

A container which is required by a pipeline stage was not found.

User action

Check that user-written programs in earlier pipeline stages use the correct containers.

AWSH

Explanation

A pipeline activity is in the wrong state.

User action

Check that user-written programs in earlier pipeline stages use the correct BTS protocols.

If your user-written programs are correct, contact your IBM support center.

AWSP

Explanation

The SOAPAction header exceeds 256 bytes in length.

User action

Check that user-written programs do not create a SOAPAction header longer than 256 bytes.

AWSR

Explanation

A container which is required by the outbound router program was not found.

User action

Check that your message adapter supplies the correct containers.

AWSS

Explanation

The service provider pipeline was not able to change the application context.

User action

- If you have attempted to change the transaction ID, check that the transaction ID that you specified is valid.
- If you have attempted to change the user ID, check that the user ID that you specified is valid.

  Check also that the original user ID under which the pipeline runs is defined as a surrogate of the target user ID (the user ID under which the transaction the message adapter and application program should run).

AWST

Explanation

There has been an input or output error in the HTTP or WebSphere MQ transport.

User action

Use CICS or WebSphere MQ messages to help you determine the cause of the problem. The error may be transient, in which case you can retry.

AWSU

Explanation

The URI used to specify a SOAP service provider is invalid.

User action

Check that user-written programs construct a valid URI.

Chapter 7. Configuring your CICS system

Before you can run SOAP applications in CICS, you must configure your CICS system. You may need to perform each of the following tasks:

1. Create a BTS repository data set
2. Define and install CICS resources

Creating the BTS repository data set

You will need to create a BTS repository data set in the following situations:

- If you use the WebSphere MQ transport
- If you use HTTP as the transport in a service requester pipeline and do not use the recommended NOCHECK option on the DEFINE PROCESS command.

Use the IDCAMS utility to create the BTS repository data set. Use the following JCL to create the data set:

```
//jobname JOB  accounting information,name
//DEFDSN   EXEC  PGM=IDCAMS,REGION=6144K
//SYSPRINT DD  SYSOUT=A
//AMSDUMP  DD  SYSOUT=A
//SYSIN    DD  *
  DELETE ('repository_data_set') PURGE  CLUSTER
  SET  MAXCC=0
  DEFINE CLUSTER (                          -
                NAME( repository_data_set )      -
                LOG(UNDO)  -
                CYL(2,1)                    -
                VOLUMES (volume)  -
                KEYS( 50 0 )                -
                INDEXED                     -
                RECORDSIZE( 4096 10240 )    -
                FREESPACE( 5 5 )            -
                SHAREOPTIONS( 2 3 )         -
                )                           -
  INDEX         (                           -
                NAME( repository_data_set.INDEX )   -
                )                           -
  DATA          (                           -
                NAME( repository_data_set.DATA )    -
                 )
/*
```

repository_data_set
is the name of the data set.

Defining CICS resources

Before you can use this feature, you must define and install the following CICS resources:

- A FILE definition for the repository data set
- A PROCESSTYPE definition for each BTS process type
- A TCPIPSERVICE definition for the port which will receive inbound SOAP requests

The feature includes sample resource definitions for these resources. For details, see “Supplied sample CICS resource definitions” on page 40

Defining the repository file

Before you can use this feature, you must define and install a FILE definition for the BTS repository data set.

Important: You must do this, even if you do not need to create the BTS repository data set itself.

1. Create a FILE definition for the repository data set. Use the following command to define the FILE:

```
DEFINE FILE(repository_file)
       GROUP(group_name)
       DSNAME(repository_data_set)
       STRINGS(20)
       DATABUFFERS(21)
       INDEXBUFFERS(20)
       ADD(YES)
       BROWSE(YES)
       DELETE(YES)
       READ(YES)
       UPDATE(YES)
```

   repository_data_set
   is the name of the data set defined in “Creating the BTS repository data set” on page 37. The sample FILE definition supplied with the feature uses the name CICSTS22.CICS.DFHWSBTS. If you do not have a BTS repository data set, you can use a dummy value for this attribute.

   You can specify your own values, or use the default values, for the other attributes.

   Note:

   - If your VSAM file uses local shared resources, then you must ensure that the LSRPOOL associated with the file is defined with a MAXKEYLENGTH of 50 bytes or more. To ensure that this is the case:
     a. Create and install an LSRPOOL definition which specifies MAXKEYLENGTH(50)
     b. Specify the LSRPOOLID attribute in your file definition. The value of this attribute must match the LSRPOOLID attribute specified in the file definition.

     The sample FILE definition supplied with the feature specifies LSRPOOL(NONE).
2. Install the definition in your CICS system
3. Enable the installed FILE definition

Defining the BTS process type

Before you can use this feature, you must define and install PROCESSTYPE definitions that point to the repository data sets used by your applications.

- Service provider applications require a PROCESSTYPE named SOAPHTTP.
- Service requester applications that contain only one activity (the root activity), can share the repository data set with service provider applications, and so can use the same PROCESSTYPE.

1. Create a PROCESSTYPE definition. Use the following command to define the PROCESSTYPE:

   ```
   DEFINE PROCESSTYPE(SOAPHTTP)
          GROUP(group_name)
          FILE(repository_file)
   ```

   repository_file
   is the name of the file defined in "Defining the repository file" on page 38

   You can specify your own values, or use the default values, for the other attributes.
2. Install the definition in your CICS system
3. Enable the installed PROCESSTYPE definition

Defining the TCPIPSERVICE

Before you can run service provider application programs, you must define and install a TCPIPSERVICE that supports the HTTP protocol.

1. Create a TCPIPSERVICE definition. Use the following command to create the TCPIPSERVICE:

   ```
   DEFINE TCPIPSERVICE(tcpip_service)
          GROUP(group_name)
          URM(DFHWBADX)
          PORTNUMBER(port_number)
          PROTOCOL(HTTP)
          TRANSACTION(CWXN)
   ```

   You can specify your own values, or use the default values, for the other attributes.

   Note: The sample TCPIPSERVICE definition supplied with the feature contains a dummy value for the PORT attribute. If you use the sample you must change this attribute to a suitable value. You should also review the other attributes to ensure that they are suitable for your purposes.
2. Install the definition in your CICS system.
3. Open the TCPIPSERVICE.

Defining CICS resources for the WebSphere MQ transport

If you plan to use the WebSphere MQ transport, you must define and install a PROCESSTYPE definition and a TRANSACTION definition.

1. Create a PROCESSTYPE definition. Use the following command to create the PROCESSTYPE:

   ```
   DEFINE PROCESSTYPE(SOAPMQ)
          GROUP(group_name)
          FILE(repository_file)
   ```
2. Create a TRANSACTION definition. Use the following command to create the TRANSACTION:

   ```
   DEFINE TRANSACTION(CWSQ)
          GROUP(group_name)
          PROGRAM(DFHWSDSQ)
          TASKDATALOC(ANY)
   ```

If you use a transaction ID other than CWSQ, it must match the transaction ID specified in the definition of the trigger process. See Chapter 8, “Configuring WebSphere MQ,” on page 43 for more information.

3. Install the definitions in your CICS system

Defining other resources

You may need to define other resources before you can use this feature:

- If you do not use autoinstall for programs, you will need to define and install the program definitions for the feature code.
- If you plan to use the sample applications, you will need to define and install the resources used by the samples.

The feature includes sample resource definitions for these other resources. For details, see “Supplied sample CICS resource definitions”

Using the supplied resource definitions

Member DFHWSCSD in library SCAVSAMP contains commands that you can use with the CSD utility program (DFHCSDUP) to define CICS resources for the feature. It contains:

- An UPGRADE command which adds resource definitions in groups DFHSOAP and DFH$SOAP to the CSD. Group DFHSOAP contains definitions which are required for the feature to function, and which you do not need to change. Group DFH$SOAP contains definitions for the supplied sample programs.
- A DEFINE command for the TCPIPSERVICE which is required for a service provider which uses the HTTP transport.
- A DEFINE command for the FILE definition for the BTS repository data set.
- ADD commands, which add all the resource definitions to list SOAPLIST

1. Review the definitions for the FILE and TCPIPSERVICE, and make any changes may be needed for your installation.
2. Run DFHCSDUP, using member DFHWSCSD as input to the job.
3. Install list SOAPLIST in your CICS system

Supplied sample CICS resource definitions

The following CICS resource definitions are supplied with the feature:

Group	Contents
SOAPUSER	Required resource definitions that may need to be modified before use
DFHSOAP	Required resource definitions that should not be modified before use
DFH$SOAP	Resource definitions for the sample programs

Resource type	Resource name	Group	Description
FILE	DFHWSBTS	SOAPUSER	BTS Repository file
TCPIPSERVICE	SOAP	SOAPUSER	HTTP port definition

Resource type	Resource name	Group	Description
PROCESSTYPE	SOAPHTTP	DFHSOAP	SOAP processtype for HTTP
PROCESSTYPE	SOAPMQ	DFHSOAP	SOAP processtype for MQ
PROGRAM	DFHWSABE	DFHSOAP	Abend Handler
PROGRAM	DFHWSAMX	DFHSOAP	Sample Application Mapper URM
PROGRAM	DFHWSDSH	DFHSOAP	SOAP Dispatcher for HTTP
PROGRAM	DFHWSDSQ	DFHSOAP	SOAP Dispatcher for MQ
PROGRAM	DFHWSHDX	DFHSOAP	Sample User Message Handler URM
PROGRAM	DFHWSPMI	DFHSOAP	Inbound Pipeline Manager
PROGRAM	DFHWSPMO	DFHSOAP	Outbound Pipeline Manager
PROGRAM	DFHWSRT	DFHSOAP	Outbound Pipeline Router
PROGRAM	DFHWSSH	DFHSOAP	SOAP Service Handler
PROGRAM	DFHWSTIH	DFHSOAP	HTTP Inbound Transport
PROGRAM	DFHWSTIQ	DFHSOAP	MQ Inbound Transport
PROGRAM	DFHWSTOH	DFHSOAP	HTTP Outbound Transport
PROGRAM	DFHWSTOQ	DFHSOAP	MQ Outbound Transport
TRANSACTION	SBOX	DFH$SOAP	Sample SOAP requester transaction
PROGRAM	DFH$WSSB	DFH$SOAP	Sample SOAP requester application
PROGRAM	DFH$WSBT	DFH$SOAP	Data-only program for sample template
MAPSET	DFH0SBM	DFH$SOAP	Sample mapset for DFH$WSSB program
DOCTEMPLATE	DFH$WSBT	DFH$SOAP	Sample template for DFH$WSSB program
PROGRAM	DFH$WSSS	DFH$SOAP	Sample SOAP provider application
PROGRAM	DFH$WSAP	DFH$SOAP	Sample Service Backend application
PROGRAM	DFH$WSDC	DFH$SOAP	Sample Service Requestor
PROGRAM	DFH$WSXL	DFH$SOAP	Sample XML tracing program

Chapter 8. Configuring WebSphere MQ

If you plan to use the WebSphere MQ transport with this feature, you must define the following objects:

- A QLOCAL object that defines the local queue used to store the SOAP messages until they are processed.
- A PROCESS object that specifies the CICS transaction which will process messages from the local queue.

The feature contains sample definitions for these resources.

1. Define the SOAP input queue. For example:

```
DEFINE                                                          -
 QLOCAL('queuename')                                            -
 DESCR('description')                                           -
 PROCESS(processname)                                           -
 INITQ('initqueue')                                             -
 TRIGGER                                                        -
 TRIGTYPE(FIRST)                                                -
 TRIGDATA('SOAP/target_program')                                -
 BOTHRESH(nnn)                                                  -
 BOQNAME('requeuename')
```

   where

   queuename
   is the name of the local queue

   processname
   is the name of the process instance that identifies the application started by the queue manager when a trigger event occurs. The name should match the name of the process specified in the definition of the SOAP input queue.

   initqueue
   is the name of the initiation queue used by your CICS system.

   target_program
   is the name of the target program.

   nnn is the number of retries that will be attempted before the message is considered failed

   requeuename
   is the name of the queue to which failed messages will be sent. This attribute is optional.

2. Define a trigger process for inbound SOAP requests. Use the following command:

```
DEFINE                                                          -
 PROCESS(processname)                                           -
 APPLTYPE(CICS)                                                 -
 APPLICID(CWSQ)                                                 -
 USERDATA('options')
```

   where:

processname
is the name of the process. The name must match the PROCESS parameter in the definition of the local queue.

options
let you specify how processes are started:

AUTH=LOCAL
Processes run under the same user ID as the CWSQ transaction

AUTH=IDENTIFY
Processes run under the user ID specified in the MQ message description (MQMD) of the incoming message

WAIT=*interval*
Specifies the time (in milliseconds) for which CWSQ should wait for more messages to arrive. The default is 60000 (one minute). If no more messages arrive within the specified interval, CWSQ terminates.

The transaction ID specified in the APPLICID parameter must match the transaction defined in "Defining CICS resources for the WebSphere MQ transport" on page 39.

Security considerations for the WebSphere MQ transport

The transport transaction (CWSQ) will be started by the trigger monitor using the same user ID as the trigger monitor. This user ID must have UPDATE authority to the input queue, the backout queue (if this is specified), and the BTS repository.

If AUTH=IDENTIFY is specified, then the user ID under which CWSQ runs must have surrogate authority to allow it to start processes on behalf of the user IDs in the messages.

The process user IDs must have UPDATE authority to the reply queue, and to the BTS repository.

Chapter 9. The sample applications

This feature includes a number of sample application programs.

DFH$WSSS

DFH$WSSS is a COBOL program that illustrates how to write a service provider application. The program contains all the logic needed to receive a SOAP request body from a client, parse the request body, and return a response to the client.

The business logic of the sample program performs an elementary lookup of a fictitious stock quotation. The client sends a body containing a `<symbol>` element that contains a stock symbol. DFH$WSSS returns a `<Quote>` element that contains a number obtained from the lookup, accompanied by `<Applid>`, `<Date>` and `<Time>` elements.

The Web Service sample application

The Web Services sample application comprises:

DFH$WSAP and DFH$WSDC
A pair of COBOL programs which, together, are functionally equivalent to the DFH$WSSS sample. The sample is structured such that:

- The back-end program, DFH$WSAP, constructs the core of the response document
- The front-end program, DFH$WSDC, adds the appropriate document element tag, and the SOAP body

As provided, the programs provide a *document/literal* service; that is:

- The **style** attribute of the operation provide by the Web Service is *document*, indicating that SOAP messages contains documents. To change the style to *rpc*, modify DFH$WSDC.
- The **use** attribute of the SOAP body is *literal*, indicating that the element tags of the SOAP messages are extended with a schema definition. To change the use to *encoded*, modify DFH$WSAP.

DFH$WSDL
Web Services Description Language (WSDL) which describes the Web Service provided by the programs as a *document/literal* service. You can use the WSDL to generate client bindings for the Web Service.

DFH$WSSB

DFH$WSSB is a COBOL program that illustrates how to write a service requester application in CICS. The program has a 3270 based user interface that allows you to submit SOAP requests to DFH$WSSS, and display the response. The DFH$WSSB program communicates with the DFH$WSSS application, using the outbound HTTP support. For more information, see “Using the DFH$WSSB sample program” on page 46.

DFH$WSXL

DFH$WSXL is a COBOL program that illustrates how the XML PARSE statement processes an XML document. It traces the XML events and data to SYSOUT.

You can initiate the program in these ways:

From a SOAP client

Call the program by sending an HTTP message containing a SOAP request, in the following form:

```
POST /CICS/CWBA/DFHWSDSH/DFH$WSXL  HTTP/1.1
Content-Type: text/xml; charset=utf-8
Content-Length: nnn
Host: host-name:port-number
SOAPAction:

SOAP message
```

From a Web browser

Call the program by entering the URI (Universal Resource Identifier[1]) in a browser. For example:

```
http://ip-address/CICS/CWBA/DFHWSDSH/DFH$WSXL
```

Type in the XML document to be parsed and press the **Submit XML document** button. The program displays information in the browser window about each event that is driven by the COBOL XML parser.

From another CICS program

Construct the XML document in the Commarea, and LINK to program DFH$WSXL. The program displays information in the CICS log about each event that is driven by the XML parser.

Using the DFH$WSSB sample program

DFH$WSSB is a COBOL program that illustrates how to write a service requester application in CICS. The program has a 3270 based user interface that allows you to submit SOAP requests to DFH$WSSS, and display the response. It is invoked by the SBOX transaction, in which the target URI (Universal Resource Identifier) is supplied as the operand of the transaction invocation.

1. Install a TRANSACTION resource definition for transaction SBOX, that points to program DFH$WSSB.
2. Install a DOCTEMPLATE resource definition. The DOCTEMPLATE must have the following attributes:

   ```
   TEMPLATENAME(SOAPBOX BODY)
   PROGRAM(DFH$SBBD)
   ```

3. Run the transaction, specifying the target as an operand. For example:

   ```
   SBOX http://ip-address/CICS/CWBA/DFHWSDSH/DFH$WSSS
   ```

   DFH$WSSB displays the following screen:

1. also known informally, although - strictly - incorrectly as the Universal Resource Locator, or URL

```
                    DFH$WSSB - CICS SOAP support demonstration


Enter the destination URI for the Stock Quote web service,
and (optionally) the URL of a proxy server.

URI . . . . .

Proxy . . . .



Enter a three-letter symbol for the stock for which a quotation is required.
(Hint - must be one of: ABC, PQR, XYZ)

         Stock symbol. . . . ___
         Quotation price . . 870 Date . . ________ Time . . ________
```

4. Complete details of the request on the screen. You can enter data in the following fields:

   URI
   Specify target resource. Initially, the field displays the target specified in the original transaction request. You can overtype the displayed value.
   - For the HTTP transport, the target is specified as the URI of the target process. For example:

     `http://ip-address/CICS/CWBA/DFHWSDSH/DFH$WSSS`

     The URI must start with the scheme name (`http://` or `https://`).
   - For the WebSphere MQ transport, the target is specified in the following form:

     `MQ://queuename/queue_manager`

     where the queue manager name is optional if the queue is local, or a remote queue definition exists. For example:

     `MQ://SAMPLE.SOAP.QUEUE/QM39`

   Proxy
   If the target resource is outside a firewall, specify the URI of a proxy server. Initially, the field displays the default proxy server, which is specified in the INITPARM system initialization parameter. For example:

   ```
   INITPARM=(DFHWBCLI='PROXY="http://myproxy.mysite.my.com"')
   ```

   Stock symbol
   Specify the three letter symbol of the stock you want to be quoted. DFH$WSSS recognizes the following values:
   - ABC
   - PQR
   - XYZ

5. Press ENTER to send the request. The following information is displayed:

   Quotation price
   Displays the stock quotation value returned by the DFH$WSSS application

   Date
   Time
   Displays the date and time returned by the DFH$WSSS application

If a SOAP fault is received, the transaction displays the contents of the fault message.

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully.

You can perform most tasks required to set up, run, and maintain your CICS system in one of these ways:

- using a 3270 emulator logged on to CICS
- using a 3270 emulator logged on to TSO
- using a 3270 emulator as an MVS™ system console

IBM® Personal Communications (Version 5.0.1 for Windows® 95, Windows 98, Windows NT® and Windows 2000; version 4.3 for OS/2®) provides 3270 emulation with accessibility features for people with disabilities. You can use this product to provide the accessibility features you need in your CICS system.

Index

T

U

W

X

Notices

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply in the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This publication could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

SupportPac
z/OS
MVS
OS/390
CICS
WebSphere

Other company, product, and service names may be trademarks or service marks of others.

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To ask questions, make comments about the functions of IBM products or systems, or to request additional publications, contact your IBM representative or your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in USA

SC34-6315-00